File No. S370-36
Order No. GC20-1818-2

Systems

IBM Virtual Machine Facility/370: CMS Command and Macro Reference

Release 6 PLC 1

This publication provides users of the Conversational Monitor System (CMS) component of IBM Virtual Machine Facility/370 with detailed reference information concerning command syntax and usage notes for:

- CMS commands
- EDIT subcommands
- DEBUG subcommands
- EXEC control statements, special variables, and built-in functions
- CMS assembler language macro instructions

PREREQUISITE PUBLICATIONS

IBM Virtual Machine Facility/370:

Terminal User's Guide, Order No. GC20-1810

CMS User's Guide, Order No. GC20-1819

Third Edition (March 1979)

| This is a major revision of, and obsoletes, GC20-1818-1 with Technical
| Newsletter GN25-0416.

| This edition applies to Release 6 PLC 1 (Program Level Change) of the IBM Virtual Machine Facility/370, and to all subsequent releases unless otherwise indicated in new editions or Technical Newsletters (TNLs).

Technical changes and additions to text and illustrations are indicated by a vertical bar to the left of the change.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370 Bibliography, Order No. GC20-0001, for the editions that are applicable and current.

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

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, VM/370 Publications, Dept. D58, Bldg. 706-2, P.O. Box 390, Poughkeepsie, New York 12602. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Preface

Use this publication as a reference manual; it contains all of the command formats, syntax rules, and operand and option descriptions for CMS commands, subcommands, and macro instructions for general users.

The _IBM Virtual Machine Facility/370: CMS User's Guide_, GC20-1819, contains tutorial information and functional descriptions of CMS commands, as well as information on using the editor, EXEC, and debugging facilities of CMS. You should be familiar with the contents of the _VM/370 CMS User's Guide_ before you attempt to use this reference manual. For most of the CMS commands described in this publication, you may find additional useful notes in the _VM/370 CMS User's Guide_.

This publication has six sections:

"Section 1. Introduction and General Concepts" describes the components of the VM/370 system and tells you how to enter CMS commands. It lists the notational conventions used in this manual, so that you can interpret the command format descriptions in Section 2. Section 1 also contains information about the CMS command search order and a summary of all the CMS commands available under VM/370, including those not for general users.

"Section 2. CMS Commands" contains complete format descriptions, and operand and option lists, for the CMS commands available to general users. Each command description contains usage notes, and lists responses and error messages (with associated return codes) produced by the command.

"Section 3. EDIT Subcommands and Macros" describes the subcommands and macros available in the environment of the CMS editor, which you can invoke with the EDIT command. Each subcommand description contains usage notes and summarizes the types of responses you might receive. Where applicable, additional information is provided for users of display terminals.

"Section 4. DEBUG Subcommands" describes the subcommands available in the debug environment of CMS. Each subcommand description contains usage notes and, where applicable, lists the responses to the subcommand.

"Section 5. EXEC Control Statements" describes the control statements, special variables, and built-in functions you can use when you create EXEC procedures to execute in CMS. The control statement descriptions contain usage notes, where applicable.

"Section 6. CMS Macro Instructions" lists the formats and operands of the CMS assembler language macro instructions you can use when you write programs to execute in CMS.

This publication also has three appendixes:

"Appendix A: Reserved Filetype Defaults" lists the filetypes that are recognized by the CMS editor and indicates the default settings that the editor supplies for logical tabs, truncation, verification, logical record length, and so on.

"Appendix B: DOS/VS Access Method Services and VSAM Functions Not Supported in CMS" lists the restrictions on the use of access method services and VSAM in the CMS/DOS environment of CMS.

"Appendix C: OS/VS Access Method Services and VSAM Functions Not Supported in CMS" lists the restrictions for OS programmers using access method services and VSAM in CMS.

Terminology

Some of the following convenience terms are used throughout this publication:

- The term "CMS/DOS" refers to the functions of CMS that become available when you issue the command:

  ```
  set dos on
  ```

  CMS/DOS is a part of the normal CMS system, and is not a separate system. Users who do not use CMS/DOS are sometimes referred to as OS users, since they use the OS simulation functions of CMS.

- The term "CMS files" refers exclusively to files that are in the 800-byte block format used by CMS file system commands. VSAM and OS data sets and DOS files are not compatible with the CMS file format, and cannot be manipulated using CMS file system commands.

The terms "disk" and "virtual disk" are used interchangeably to indicate disks that are in your CMS virtual machine configuration. Where necessary, a distinction is made between the CMS-formatted disks and disks in OS or DOS format.

The following terms in this publication refer to the indicated support devices:

- "2305" refers to IBM 2305 Fixed Head Storage, Models 1 and 2.
- "270x" refers to IBM 2701, 2702, and 2703 Transmission Control Units or the Integrated Communications Adapter (ICA) on the System/370 Model 135.
- "3270" refers to a series of display devices, namely, the IBM 3275, 3276, 3277, and 3278 Display Stations. A specific device type is used only when a distinction is required between device types.

  Information about display terminal usage also applies to the IBM 3138, 3148, and 3158 Display Consoles when used in display mode, unless otherwise noted.

  Any information pertaining to the IBM 3284 or 3286 Printer also pertains to the IBM 3287, 3288, and 3289 printers, unless otherwise noted.
- "3330" refers to the IBM 3330 Disk Storage Models 1, 2, or 11; and the 3350 Direct Access Storage operating in 3330/3333 Model 1 or 3330/3333 Model 11 compatibility mode.
- "3340" refers to the IBM 3340 Disk Storage, Models A2, B1, and B2, and the 3344 Direct Access Storage Model B2.
- "3350" refers to the IBM 3350 Direct Access Storage Models A2 and B2 in native mode.
- "3704", "3705", or "3704/3705" refers to IBM 3704 and 3705 Communications Controllers.
- "3705" refers to the 3705 I and the 3705 II unless otherwise noted.
- "2741" refers to the IBM 2741 and the 3767, unless otherwise specified.
- | "3066" refers to the IBM 3066 System Console.

For a glossary of VM/370 terms, see the _IBM Virtual Machine Facility/370: Glossary and Master Index_, GC20-1813.

PREREQUISITE PUBLICATIONS

In addition to the _VM/370 CMS User's Guide_, prerequisite information is contained in the following publications:

- For information about the terminal that you are using, including procedures for gaining access to the VM/370 system and logging on, see the _IBM Virtual Machine Facility/370: Terminal User's Guide_, GC20-1810.
- If you are using an IBM 3767 Communications Terminal, the _IBM 3767 Operator's Guide_, GA18-2000, is a prerequisite.
- The CP commands that are available to you as a general user are described in _IBM Virtual Machine Facility/370: CP Command Reference for General Users_, GC20-1820.

For additional tutorial information on using CMS, you may want to use _CMS for Programmers - A Primer_, SR20-4438.

If you are going to use an IBM Program Product compiler under CMS, you should have available the appropriate program product documentation. These publications are listed in _IBM Virtual Machine Facility/370: Introduction GC20-1800_.

COREQUISITE PUBLICATIONS

The _IBM Virtual Machine Facility/370: System Messages_, GC20-1808, describes all of the error messages and system responses produced by the CMS commands and EDIT and DEBUG subcommands referenced in this publication. It also lists the error messages issued by the EXEC processor during execution of your EXEC procedures.

If you are alternating between CMS and other operating systems in virtual machines running under VM/370, you should consult _IBM Virtual Machine Facility/370: Operating Systems in a Virtual Machine_, GC20-1821.

SUPPLEMENTAL PUBLICATIONS

For general information about the VM/370 system, see the publications IBM Virtual Machine Facility/370: Introduction, GC20-1800, and VM/370 Features Supplement, GC20-1757.

Additional descriptions of various CMS functions and commands which are normally used by system support personnel are described in

IBM Virtual Machine Facility/370:

System Programmer's Guide, GC20-1807

Operator's Guide, GC20-1806

Planning and System Generation Guide, GC20-1801

Information on IPCS commands, which are invoked under CMS, is contained in IBM Virtual Machine Facility/370: Interactive Problem Control System (IPCS) User's Guide, GC20-1823.

Details on the CMS CPEREP, a command used to generate output reports from VM/370 error recording records, are contained in:

IBM Virtual Machine Facility/370: OLTSEP and Error Recording Guide, GC20-1809.

For more details on the operands used with CPEREP, refer to:

OS/VS, DOS/VSE, VM/370 Environmental Recording, Editing, and Printing (EREP) Program, GC28-0772.

For messages issued by CMS CPEREP, see:

OS/VS, DOS/VSE, VM/370 EREP Messages, GC38-1045.

For VM/370 Users

There are three publications available as ready reference material when you use VM/370 and CMS. They are:

IBM Virtual Machine Facility/370:

Quick Guide for Users, GX20-1926

Commands (General User), GX20-1961.

Commands (other than General User), GX20-1995.

If you are going to use the Remote Spooling Communications Subsystem, see the IBM Virtual Machine Facility/370: Remote Spcoling Communications Subsystem (RSCS) User's Guide, GC20-1816.

Assembler language programmers may find information about the VM/370 assembler in OS/VS, DOS/VS, and VM/370 Assembler Language, Order No. GC33-4010, and OS/VS and VM/370 Assembler Programmer's Guide, GC33-4021.

For VSAM and Access Method Services Users

CMS support of Access Method Services is based on DOS/VS Access Method Services. The control statements that you can use are described in DOS/VS Access Method Services User's Guide, GC33-5382. The VM/370: CMS User's Guide contains details on how to use this support. Error messages produced by the Access Method Services program, and return codes and reason codes are listed in DOS/VS Messages, GC33-5379.

For a detailed description of DOS/VS VSAM macros and macro parameters, refer to the DOS/VS Supervisor and I/O Macros, GC33-5373. For information on OS/VS VSAM macros, refer to OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide, GC26-3818.

For CMS/DOS Users

The CMS ESERV command invokes the DOS/VS ESERV program, and uses, as input, the control statements that you would use in DOS/VS. These control statements are described in Guide to the DOS/VS Assembler, GC33-4024.

Linkage editor control statements, used when invoking the DOS/VS linkage editor under CMS/DOS, are described in DOS/VS System Control Statements, GC33-5376.

Batch DL/I application programs can be written and tested in the CMS/DOS environment. See VM/370 CMS User's Guide, GC20-1819, and DL/I DOS/VS General Information, GH20-1246, for details.

Contents

The entries in this Table of Contents are accumulative and reflect the addition of the VM/370 Basic System Extensions Program Product, Program Number 5748-XX8.

3278-2A DISPLAY CONSOLE

New: Program Feature

The CMS editor now supports the 3278 Model 2A Display Console which is a 20-line display console. "Section 3. EDIT Subcommands and Macros" is modified to reflect this support.

MISCELLANEOUS

Changed: Documentation

Technical corrections and editorial changes have been made throughout this publication.

Summary of Amendments
for GC20-1818-1
as updated by GN25-0416
Release 5 PLC 1

DOS/VS RELEASE 34 SUPPORTED

New: Program Feature

CMS/DOS supports DOS/VS Release 34. This support includes a new operand of the SET command and a new operand of the QUERY command. SET DOSLNCNT allows the user to set the number of SYSLST lines per page. QUERY DOSLNCNT displays the current number of SYSLST lines per page. These new operands are described in "Section 2. CMS Commands."

MISCELLANEOUS

Changed: Programming and Documentation

Minor technical and editorial changes have been made to clarify the text.

IBM VM/370 ATTACHED PROCESSOR SUPPORT

New: Programming and Hardware Changes

VM/370 support for the IBM System/370 Attached Processor is now available for the System/370 Model 158 and 168 processors. Modifications to the program are documented, such as the use of hardware prefixing, which allows each processor to have its own PSA, and a series of locks, which provide the necessary controls.

IBM VM/370 SUPPORT FOR THE DEDICATED 3850 MASS STORAGE SYSTEM

New: Programming

VM/370 supports the 3850 Mass Storage System as a dedicated device. As many as four virtual machines may be concurrently running OS/VS1 or OS/VS2, each of which (with MSS support) can control an interface with a common 3850 Mass Storage System.

Dedicated MSS support permits an installation to generate the system, test, and convert to an MSS environment while concurrently running non-MSS production.

VM/370 SUPPORTS THE 3270 DISPLAY DEVICES

Changed: Programming and Documentation

VM/370 now supports 3270 display devices. The term 3270 now refers to the IBM 3275, 3276, 3277, and 3278 Display Stations. It also applies to the IBM 3138, 3148, and 3158 Display Consoles, when used in display mode. Information pertaining to the IBM 3284 or 3286 Printers also pertains to the IBM 3287, 3288, and 3289 Printers.

VM/370 SUPPORTS OS/VS EREP (IFCEREP1)

Changed: Programming and Documentation

The CPEREP command now uses all edit and format operands that are available to OS/VS EREP. Because of VM/370's compatibility with OS/VS EREP, VM/370 relies on existing OS/VS EREP documentation. Therefore, VM/370 no longer publishes the following:

IBM Virtual Machine Facility/370:

Environmental Error Recording, Editing, and Printing (_EREP_) _Program_, Order No. GC29-8300

Environmental Error Recording, Editing, and Printing (_EREP_) _Program Logic_, Order No. SY25-7701

Documentation of the interface to OS/VS EREP and the interface to the VM/370 error recording cylinders is contained in:

IBM Virtual Machine Facility/370:

OLTSEP and Error Recording Guide, Order No. GC20-1809

Service Routines Program Logic, Order No. SY20-0882

VM/370 publications contain referrals to OS/VS publications where required.

The following areas in this publication reflect the changes to EREP documentation:

Preface
Section 1. Introduction and General Concepts

MISCELLANEOUS

Changed: Programming and Documentation

Minor technical and editorial changes have been made to clarify the text.

Section 1. Introduction and General Concepts

Virtual Machine Facility/370 (VM/370) is a system control program (SCP) that controls "virtual machines." A virtual machine is the functional equivalent of a real machine, but where the real machine has lights to show status, and buttons and switches on the real system console to control it, the virtual machine has a virtual system console to display status and a command language to start operations and control them. The virtual system console is your terminal; there are three command languages, which correspond roughly to the four components of the VM/370 system:

- The Control Program (CP) controls the resources of the real machine; that is, the physical machine in your computer room. The CP commands are described in _VM/370 CP Command Reference for General Users_.

- The Remote Spooling Communications Subsystem (RSCS) is a subsystem designed to supervise transmission of files across a teleprocessing network controlled by CP. For information about RSCS, see the _VM/370 Remote Spooling Communications Subsystem (RSCS) User's Guide_.

- The Conversational Monitor System (CMS) is a conversational operating system designed to run under CP. All of the CMS commands for general use, and the subcommands and macros that you can use in the CMS environment, are described in this publication.

- The Interactive Problem Control System (IPCS) provides system programmers and installation support personnel with VM/370 problem analysis and management facilities, including problem report creation, problem tracking, and CP abend dump analysis. IPCS runs in the CMS command environment; for details, see _VM/370 IPCS User's Guide_.

Except for IPCS, each of the components of VM/370 has a unique "command environment" which must be active in order for a command to be accepted. For CMS users, the two basic command environments are the CP command environment and the CMS command environment. By default, CP commands are acceptable input in the CMS command environment; if you enter a CP command, it is executed by CP, but control returns to the CMS environment.

The CMS Environment

The CMS command language allows you to create, modify, debug, and, in general, manipulate a system of files.

The OS/VS Assembler and many OS/VS and DOS/VSE Language processors can be executed under CMS. For example, the OS VS BASIC, FORTRAN IV (G1), COBOL/ and PL/I compilers, as well as the DOS PL/I and COBOL compilers, can execute under CMS. You can find a complete list cf language processors that can be executed under CMS in the _VM/370 Introduction_. CMS invokes the assembler and the compilers when you issue the appropriate CMS commands. The ASSEMBLE command is described in this manual; the supported compiler commands are described in the appropriate program product publications.

CMS commands allow you to read cards from a virtual card reader, punch cards to a virtual card punch, and print records on a virtual printer. Many commands are provided to help you manipulate your virtual disks and files. The CMS commands are described in "Section 2. CMS Commands."

A special set of CMS commands becomes available to you when you issue the command:

```
set dos on
```

These commands, called CMS/DOS commands, simulate various functions of the Disk Operating System (DOS) in your CMS virtual machine. When the CMS/DOS environment is active, the CMS/DOS commands are an integral part of the CMS command language; they are listed alphabetically among the other CMS commands in "Section 2. CMS Commands."

The EDIT command places your virtual machine in the EDIT subcommand environment. In this environment you can use the CMS editor to create and modify files. In the EDIT subcommand environment, you can place your virtual machine in either of two modes, edit mode or input mode. Edit mode lets you modify a file; input mode lets you create or add to a file. The subcommands available to you in the EDIT subcommand environment are described in "Section 3. EDIT Subcommands and Macros."

The DEBUG command places your virtual machine in the DEBUG subcommand environment. In this environment you can issue commands to display registers and storage, specify breakpoints (address instruction stops), display the contents of control words, and so on. The DEBUG subcommands are described in "Section 4. DEBUG Subcommands."

The EXEC command executes CMS command procedures, called EXEC files. You can create EXEC files consisting of CMS and CP commands and EXEC control statements. The EXEC facility also has a symbolic capability; by manipulating variable symbols within an EXEC file, you can control the execution of the procedure. These procedures are usually created in the edit environment. The EXEC control statements, variable symbols, and built-in functions are described in "Section 5. EXEC Control Statements."

You can use the CMS assembler language macros when you write assembler language programs to execute in the CMS environment. Descriptions of these macros are contained in "Section 6. CMS Macro Instructions."

The HELP format words are used to create HELP 'text' information for user-defined commands, EXECs, and messages. The function, formats, and operands of the HELP facility format words are described in "Section 7. HELP Format Words."

Entering CMS Commands

A CMS command consists of a command name, usually followed by one or more positional operands and, in many cases, by an option list. CMS commands and EDIT and DEBUG subcommands described in this publication are shown in the format:

```
| command name | [operands...] [(options...[)]] |
```

You must use one or more blanks to separate each entry in the command line unless otherwise indicated. For an explanation of the special symbols used to describe the command syntax, see "Notational Conventions."

The Command Name

The command name is an alphameric symbol of one to eight characters. In general, the names are based on verbs that describe the function ycu want the system to perform. For example, you may want to find out information concerning your CMS files. In this case, you would use the LISTFILE command.

The Command Operands

The command operands are keywords and/or positional operands of one to eight, and in a few cases, one to seven alphameric characters each. The operands specify the information on which the system operates when it performs the command function.

You must write the operands in the order in which they appear in the command formats in "Section 2. CMS Commands," unless otherwise specified. When you are using CMS, blanks may optionally be used to separate the last operand from the option list. CMS recognizes a left parenthesis "(" as the beginning of an option list; it does not have to be preceded by a blank.

The Command Options

The command options are keywords used to contrcl the execution of the command. The command formats in "Section 2. CMS Commands" show all the options for each CMS command.

The option list must be preceded by a left parenthesis; the closing parenthesis is not necessary.

For most commands, if conflicting or duplicate options are entered, the last option entered is the option in effect for the command. Exceptions to this rule are noted where applicable.

Comments in CMS Command Lines

If you want to write comments with CMS commands, you enter them following the closing parenthesis of the option list. The only exception to this rule is the ERASE command, for which comments are not allowed.

You can also enter comments on your conscle by using the CP * command.

Character Set Usage

CMS commands may be entered using a combination of characters from six different character sets. The contents of each cf the character sets is shown in Figure 1.

Character Set	Names	Symbols
Separator	Blank	
National	Dollar Sign	$
	Pound Sign	#
	At Sign	@
Alphabetic	Uppercase	A – Z
	Lowercase	a – z
Numeric	Numeric	0 – 9
Alphameric	National	$, #, @
	Alphabetic	A – Z
		a – z
	Numeric	0 – 9
Special		All other characters

Figure 1. Character Sets and Their Contents

Notational Conventions

The notation used to define the command syntax in this publication is:

- Truncations and Abbreviations of Commands

  Where truncation of a command name is permitted, the shortest acceptable version of the command is represented by uppercase letters. (Remember, however, that CMS commands can be entered with any combination of uppercase and lowercase letters.) The following example shows the format specification for the FILEDEF command.

  ```
  FIledef
  ```

  This format means that FI, FIL, FILE, FILED, FILEDE, and FILEDEF are all valid specifications for this command name.

  Operands and options are specified in the same manner. Where truncation is permitted, the shortest acceptable version of the operand or option is represented by uppercase letters in the command format box. If no minimum truncation is noted, the entire word (represented by all uppercase letters) must be entered.

  Abbreviations are shorter forms of command operands and options. Abbreviations for operands and options are shown in the description of the individual operands and options that follow the format box. For example, the abbreviation for MEMBER in the PRINT command is MEM. Only these two forms are valid and no truncations are allowed. The format box contains

  ```
  MEMBER { name }
         {  *   }
  ```

  and the description that follows the format box is

  ```
  MEMBER { name }
  MEM    {  *   }
  ```

- The following symbols are used to define the command format and should never be typed when the actual command is entered.

underscore	_
braces	{ }
brackets	[ ]
ellipsis	...

- Uppercase letters and words, and the following symbols, should be entered as specified in the format box.

asterisk	*
comma	,
hyphen	-
equal sign	=
parentheses	( )
period	.
colon	:

- The abbreviations "fn", "ft", and "fm" refer to filename, filetype, and filemode, respectively. The combination "fn ft [fm]" is also called the file identifier or fileid.

  When a command format box shows the characters, fn ft fm or fileid and they are not enclosed by brackets or braces, it indicates that a CMS file identifier must be entered. If an asterisk (*) appears beneath fn, ft, or fm, it indicates that an asterisk may be coded in that position of the fileid. The operand description describes the usage of the *.

- Lowercase letters, words, and symbols that appear in the command format box represent variables for which specific information should be substituted. For example, "fn ft fm" indicates that file identifiers such as "MYFILE EXEC A1" should be entered.

- Choices are represented in the command format boxes by stacking.

  A
  B
  C

- An underscore indicates an assumed default option. If an underscored choice is selected, it need not be specified when the command is entered.

  Example
  The representation

  A
  B
  C

  indicates that either A, B, or C may be selected. However, if B is selected, it need not be specified. Or, if none is entered, B is assumed.

- The use of braces denotes choices, one of which must be selected.

  Example
  The representation

  ```
  ⎧ A ⎫
  ⎨ B ⎬
  ⎩ C ⎭
  ```

  indicates that you must specify either A, or B, or C. If a list of choices is enclosed by neither brackets or braces, it is to be treated as if enclosed by braces.

- The use of brackets denotes choices, one of which may be selected.

  Example:
  The representation

  ```
  ┌   ┐
  │ A │
  │ B │
  │ C │
  └   ┘
  ```

  indicates that you may enter A, B, or C, or you may omit the field.

- In instances where there are nested braces or brackets on the text lines, the following rule applies: nested operand selection is dependent upon the selection of the operand of a higher level of nesting.

  Example:

  ```
   Level 1   Level 2   Level 3
  [filename [filetype [filemode]]]
  ```

  where the highest level of nesting is the operand that is enclosed in only one pair of brackets and the lowest level of nesting is the operand that is enclosed by the maximum number of brackets. Thus, in the previous example, the user has the option of selecting a file by filename only or filename filetype only or by filename filetype filemode. The user cannot select filetype alone because filetype is nested within filename and our rule states: the higher level of nesting must be selected in order to select the next level (lower level) operand. The same is true if the user wants to select filemode; filename and filetype must also be selected.

- An ellipsis indicates that the preceding item or group of items may be repeated more than once in succession.

  Example
  The representation

  ```
  (options...)
  ```

  indicates that more than one option may be coded within the parentheses.

CMS Command Search Order

When you enter a command name at the terminal, CMS begins searching for the command of that name. Once a match is found, the search stops. The search order is:

1. EXEC file on any currently accessed disk. CMS uses the standard search order (A through Z.)

2. Valid abbreviation or truncation for an EXEC file on any currently accessed disk, according to current SYNONYM file definitions in effect.

3. CMS command that has already been loaded into the transient area.

   The commands that execute in the transient area are:

   ACCESS
   ASSGN
   COMPARE
   DISK
   DLBL
   FILEDEF
   GENDIRT
   GLOBAL
   HELP
   LISTFILE
   MODMAP
   OPTION
   PRINT
   PUNCH
   QUERY
   READCARD
   RELEASE
   RENAME
   SET
   SVCTRACE
   SYNONYM
   TAPE
   TYPE

4. CMS nucleus-resident command. The nucleus-resident CMS commands are:

   CP
   DEBUG
   ERASE
   FETCH
   GENMOD
   INCLUDE
   LOAD
   LOADMOD
   START
   STATE
   STATEW

5. Command module on any currently accessed disk. (All the remaining CMS commands are disk-resident and execute in the user area.)

6. Valid abbreviation or truncation for nucleus-resident or transient area command module.

7. Valid abbreviation or truncation for disk-resident command.

Figure 2 shows a basic description of the command search order; you can find complete details in the VM/370 System Programmer's Guide.

CMS Command Summary

Figures 3 and 4 contain alphabetical lists of the CMS commands and the functions performed by each. Figure 3 lists those commands that are available for general use; Figure 4 lists the commands used by system programmers and system support personnel who are responsible for generating, maintaining, and updating VM/370. Unless otherwise noted, CMS commands are described in this manual. In these figures, the "Code" column indicates, for those commands not described in this manual, the reference source for that command:

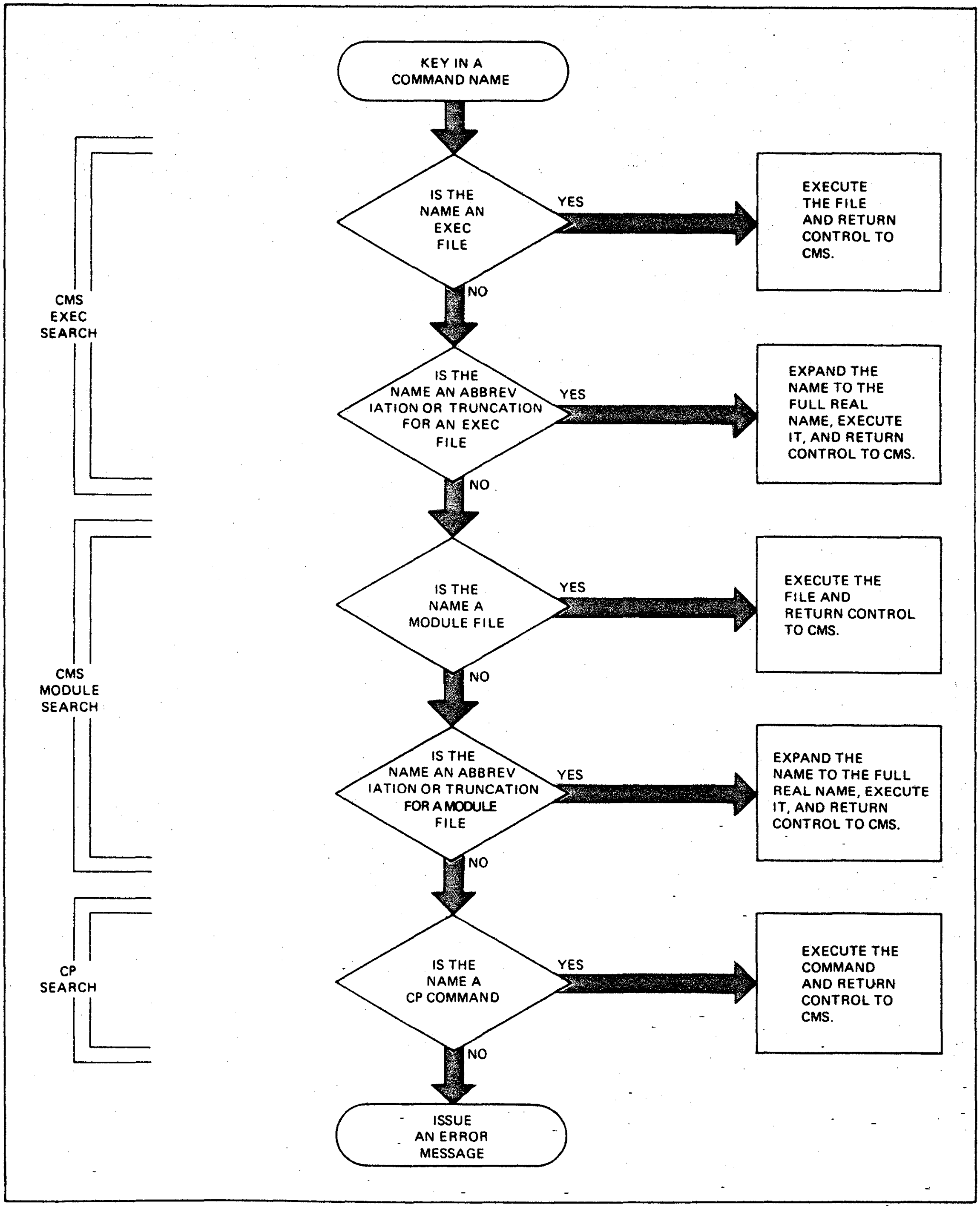

KEY IN A COMMAND NAME
CMS EXEC SEARCH
IS THE NAME AN EXEC FILE
YES
EXECUTE THE FILE AND RETURN CONTROL TO CMS.
NO
IS THE NAME AN ABBREV IATION OR TRUNCATION FOR AN EXEC FILE
YES
EXPAND THE NAME TO THE FULL REAL NAME, EXECUTE IT, AND RETURN CONTROL TO CMS.
NO
CMS MODULE SEARCH
IS THE NAME A MODULE FILE
YES
EXECUTE THE FILE AND RETURN CONTROL TO CMS.
NO
IS THE NAME AN ABBREV IATION OR TRUNCATION FOR A MODULE FILE
YES
EXPAND THE NAME TO THE FULL REAL NAME, EXECUTE IT, AND RETURN CONTROL TO CMS.
NO
CP SEARCH
IS THE NAME A CP COMMAND
YES
EXECUTE THE COMMAND AND RETURN CONTROL TO CMS.
NO
ISSUE AN ERROR MESSAGE

Figure 2. How CMS Searches for the Command to Execute

Code	Meaning
DOS PP	indicates that this command invokes a DOS Program Product, available from IBM for a license fee.
EREP	indicates that this command is described in the VM/370 OLTSEP and Error Recording Guide; further details on the operands used by this command are contained in the OS/VS, DOS/VSE, VM/370 Environmental Recording, Editing, and Printing (EREP) Program.
IPCS	indicates that this command is a part of the Interactive Problem Control System (IPCS), and is invoked under CMS. It is described in the VM/370 Interactive Problem Control System (IPCS) User's Guide.
Op Gd	indicates that this command is described in the VM/370 Operator's Guide.
OS PP	indicates that this command invokes an OS Program Product, available from IBM for a license fee.
SCRIPT	indicates that this command invokes a text processor that is an IBM Installed User Program, available from IBM for a license fee.
SPG	indicates that this command is described in the VM/370 System Programmer's Guide.
SYSGEN	indicates that this command is described in the VM/370 Planning and System Generation Guide.

Note: If a CMS command is described in this manual, but is also repeated in other VM/370 publications, the chart does not refer to those other publications.

You can enter CMS commands when you are running CMS in your virtual machine, the terminal is idle, and the virtual machine is receptive for input. However, if CMS is processing a previously entered command and your typewriter terminal keyboard is locked, you must signal your virtual machine via an attention interruption. The system acknowledges the interruption by unlocking the keyboard. Now you can enter commands.

If your terminal is a display device, there is no problem of entering commands while the virtual machine is busy as its keyboard remains unlocked for additional command input. Note that in these circumstances the command you enter is stacked and is not executed until the command that is currently being executed completes. If more commands are entered than can be handled by CP, a NOT ACCEPTED message is displayed at the display terminal.

In addition to the commands listed in Figures 3 and 4, there are seven commands called Immediate commands which are handled in a different manner from the others. They may be entered while another command is being executed by pressing the Attention key (or its equivalent), and they are executed immediately. The Immediate commands are:

- HB - Halt batch execution
- HO - Halt tracing
- HT - Halt typing
- HX Halt execution
- RO - Resume tracing
- RT - Resume typing
- SO - Suspend tracing

Command	Code	Usage
ACCESS		Identify direct access space to a CMS virtual machine, create extensions and relate the disk space to a logical directory.
AMSERV		Invoke access method services utility functions to create, alter, list, copy, delete, import, or export VSAM catalogs and data sets.
ASSEMBLE		Assemble assembler language source code.
ASSGN		Assign or unassign a CMS/DOS system or programmer logical unit for a virtual I/O device.
CMSBATCH		Invoke the CMS batch facility.
COBOL	OS PP	Compile OS ANS Version 4 or OS/VS COBOL source code.
COMPARE		Compare records in CMS disk files.
CONVERT	OS PP	Convert free form FORTRAN statements to fixed form.
COPYFILE		Copy CMS disk files according to specifications.
CP		Enter CP commands from the CMS environment.
CPEREP	EREP	Format and edit system error records for output.
DDR		Perform backup, restore, and copy operations for disks.
DEBUG		Enter DEBUG subcommand environment.
DISK		Perform disk-to-card and card-to-disk operations for CMS files.
DLBL		Define a DOS filename or VSAM ddname and relate that name to a disk file.
DOSLIB		Delete, compact, or list information about the phases of a CMS/DOS phase library.
DOSLKED		Link-edit CMS text decks or object modules from a DOS/VSE relocatable library and place them in executable form in a CMS/DOS phase library.
DOSPLI	DOS PP	Compile DOS PL/I source code under CMS/DOS.
DSERV		Display information contained in the DOS/VSE core image, relocatable, source, procedure, and transient directories.

Figure 3. CMS Command Summary (Part 1 of 4)

Command	Code	Usage
EDIT		Invoke the CMS editor to create or modify a disk file.
ERASE		Delete CMS disk files.
ESERV		Display, punch or print an edited (compressed) macro from a DOS/VSE source statement library (E sublibrary).
EXEC		Execute special procedures made up of frequently used sequences of commands.
FCOBOL	DOS PP	Compile DOS/VS COBOL source code under CMS/DOS.
FETCH		Fetch a CMS/DOS or DOS/VSE executable phase.
FILEDEF		Define an OS ddname and relate that ddname to any device supported by CMS.
FORMAT		Prepare disks in CMS fixed block format.
FORTGI	OS PP	Compile FORTRAN source code using the G1 compiler.
FORTHX	OS PP	Compile FORTRAN source code using the H-extended compiler.
GENDIRT		Fill in auxiliary module directories.
GENMOD		Generate nonrelocatable CMS files (MODULE files).
GLOBAL		Identify specific CMS libraries to be searched for macros, copy files, missing subroutines, or DOS executable phases.
GOFORT	OS PP	Compile FORTRAN source code and execute the program using the FORTRAN Code and Go compiler.
HELP		Display information about CP, CMS, or user commands and messages.
INCLUDE		Bring additional TEXT files into storage and establish linkages.
LABELDEF		Specify standard HDR1 and EOF1 tape label description information for CMS, CMS/DOS, and OS simulation.
LISTDS		List information about data sets and space allocation on OS, DOS, and VSAM disks.
LISTFILE		List information about CMS disk files.
LISTIO		Display information concerning CMS/DOS system and programmer logical units.
LOAD		Bring TEXT files into storage for execution.
LOADMOD		Bring a single MODULE file into storage.
MACLIB		Create or modify CMS macro libraries.

Figure 3. CMS Command Summary (Part 2 of 4)

Command	Code	Usage
MODMAP		Display the load map of a MODULE file.
MOVEFILE		Move data from one device to another device of the same or a different type.
OPTION		Change the DOS COBOL compiler (FCOBOL) options that are in effect for the current terminal session.
PLIC	OS PP	Compile and execute PL/I source code using the PL/I Checkout Compiler.
PLICR	OS PP	Execute the PL/I object code generated by the OS PL/I Checkout Compiler.
PLIOPT	OS PP	Compile PL/I source code using the OS PL/I Optimizing Compiler.
PRINT		Spool a specified CMS file to the virtual printer.
PSERV		Copy a procedure from the DOS/VSE procedure library onto a CMS disk, display the procedure at the terminal, or spool the procedure to the virtual punch or printer.
PUNCH		Spool a copy of a CMS file to the virtual punch.
QUERY		Request information about a CMS virtual machine.
READCARD		Read data from spooled card input device.
RELEASE		Make a disk and its directory inaccessible to a CMS virtual machine.
RENAME		Change the name of a CMS file or files.
RSERV		Copy a DOS/VSE relocatable module onto a CMS disk, display it at the terminal, or spool a copy to the virtual punch or printer.
RUN		Initiate series of functions to be performed on a source, MODULE, TEXT, or EXEC file.
SCRIPT	SCRIPT	Format and print documents according to embedded SCRIPT control words in the document file.
SET		Establish, set, or reset CMS virtual machine characteristics.

Figure 3. CMS Command Summary (Part 3 of 4)

Command	Code	Usage
SORT		Arrange a specified file in ascending order according to sort fields in the data records.
SSERV		Copy a DOS/VSE source statement book onto a CMS disk, display it at the terminal, or spool a copy to the virtual punch or printer.
START		Begin execution of programs previously loaded (OS and CMS) or fetched (CMS/DOS).
STATE		Verify the existence of a CMS disk file.
STATEW		Verify a file on a read/write CMS disk.
SVCTRACE		Record information about supervisor calls.
SYNONYM		Invoke a table containing synonyms you have created for CMS and user-written commands.
TAPE		Perform tape-to-disk and disk-to-tape operations for CMS files, position tapes, and display or write VOL1 labels.
TAPEMAC		Create CMS MACLIB libraries directly from an IEHMOVE-created partitioned data set on tape.
TAPPDS		Load OS partitioned data set (PDS) files or card image files from tape to disk.
TESTCOB	OS PP	Invoke the OS COBOL Interactive Debug Program.
TESTFORT	OS PP	Invoke the FORTRAN Interactive Debug Program.
TXTLIB		Generate and modify text libraries.
TYPE		Display all or part of a CMS file at the terminal.
UPDATE		Make changes in a program source file as defined by control cards in a control file.
VSAPL	OS PP	Invoke VS APL interface in CMS.
VSBASIC	OS PP	Compile and execute VS BASIC programs under CMS.
VSBUTIL	OS PP	Convert BASIC 1.2 data files to VS BASIC format.

Figure 3. CMS Command Summary (Part 4 of 4)

Command	Code	Usage
ASM3705	SYSGEN	Assemble 370x source code.
ASMGEND	SYSGEN	Regenerate the VM/370 assembler command modules.
CMSGEND	SYSGEN	Generate a new CMS disk-resident module from updated TEXT files.
CMSXGEN	SYSGEN	Generate the CMSSEG discontiguous saved segment.
CPEREP	EREP	Format and edit system error records for output.
DIRECT	Op Gd	Set up VM/370 directory entries.
DOSGEN	SYSGEN	Load and save the CMSDOS shared segment.
DUMPSCAN	IPCS	Provide interactive analysis of CP abend dumps.
GEN3705	SYSGEN	Generate an EXEC file that assembles and link-edits the 370x control program.
GENERATE	SYSGEN	Update VM/370 or the VM/370 directory, or generate a new standalone copy of a service program.
LKED	SYSGEN	Link-edit the 370x control program.
NCPDUMP	OP Gd, SPG	Process CP spool reader files created by 370x dumping operations.
PRB	IPCS	Update IPCS problem status.
PROB	IPCS	Enter a problem report in IPCS.
SAVENCP	SYSGEN, SPG	Read 370x control program load into virtual storage and save an image on a CP-owned disk.
SETKEY	SPG	Assign storage protect keys to storage assigned to named systems.
STAT	IPCS	Display the status of reported system problems.
VMFBLD	SYSGEN	Generate and/or update VM/370 using the PLC tape.
VMFDOS	SYSGEN	Create CMS files for DOS modules from DOS library distribution tape or SYSIN tape.
VMFDUMP	Op Gd, IPCS	Format and print system abend dumps; under IPCS, create a problem report.
VMFLOAD	SYSGEN	Generate a new CP, CMS or RSCS module.
VSAMPP	SYSGEN	Load and save the CMSVSAM, CMSAMS, and CMSBAM segments.
ZAP	Op Gd, SPG	Modify or dump LOADLIB, TXTLIB, or MODULE files.

Figure 4. CMS Commands for System Programmers

Section 2. CMS Commands

This section contains reference information for the CMS commands used by general users. Each command description indicates the format, operands and options, and error messages and return codes issued by the command. Usage notes are provided, where applicable.

The formats of the DEBUG, EDIT, and EXEC commands are also listed; for details on the EDIT or DEBUG subcommands or EXEC control statements, see:

- "Section 3. EDIT Subcommands and Macros"
- "Section 4. DEBUG Subcommands"
- "Section 5. EXEC Control Statements"

For more detailed usage information on CMS commands, see the *VM/370 CMS User's Guide*.

ACCESS

Use the ACCESS command to identify a disk to CMS, establish a filemode letter for the files on the disk, and set up a file directory in storage. The specifications of the ACCESS command determine the entries in the user file directory. The format of the ACCESS command is:

```
┌──────────────────────────────────────────────────────────────────────┐
|          |  ┌                                                 ┐      |
| ACcess   |  | cuu mode[/ext [fn [ft [fm]]]]  [(NOPROF [)]]    |      |
|          |  | 191  A          *   *   *                       |      |
|          |  |                                                 |      |
|          |  | cuu mode (ERASE [)]                             |      |
|          |  |                                                 |      |
|          |  |  (NODISK [)]                                    |      |
|          |  └                                                 ┘      |
└──────────────────────────────────────────────────────────────────────┘
```

where:

cuu makes the disk at the specified virtual device address available. The default value is 191.

Valid addresses are 001 through 5FF for a virtual machine in basic control mode, and 001 through FFF for a virtual machine in extended control mode.

mode assigns a one-character filemode letter to all files on the disk being accessed. This field must be specified if cuu is specified. The default value is A.

ext indicates the mode of the parent disk. Files on the disk being accessed (cuu) are logically associated with files cn the parent disk; the disk at cuu is considered a read-only extension. A blank must not precede or follow the diagonal (/).

fn [ft [fm]]
defines a subset of the files on the specified disk. Only the specified files are included in the user file directory and only those files can be read. An asterisk coded in any of these fields indicates all filenames, filetypes, or filemode numbers (except 0) are to be included. (See Usage Notes 3 and 4.) If a filemode is specified, it must be specified as a letter and a number. For OS and DOS disk access restrictions, see Usage Note 9.

Options:

NOPROF suppresses execution of a PROFILE EXEC file. This opticn is valid only if the ACCESS command is the first command entered after you IPL CMS. On subsequent ACCESS commands, the NOPROF option is ignored.

ERASE specifies that you want to erase all of the files on the specified disk. This option is only valid for read/write disks. (See Usage Note 7.)

NODISK lets you gain access to the CMS cperating system with no disks accessed except the system disk (S-disk) and its extensions. This option is only valid if the ACCESS command is the first command you enter after you IPL CMS.

Usage Notes

1. If you have disk addresses 190, 191, 192, and 19E defined in the VM/370 directory, or if they are defined before you IPL CMS, these disks are accessed as the S-, A-, D-, and Y-disks respectively. You must issue explicit ACCESS commands to access any other disks you wish to use following an IPL of the CMS system. Ordinarily, you have access only to files with a filemode number of 2 on the system disk.

   When ACCESS is the first command issued after an IPL of the CMS system, the A-disk is not automatically defined. Another ACCESS command must be issued to define the A-disk.

2. Each CMS disk has associated with it a file directory, which contains an entry for every CMS file on the disk. The user file directory created in storage by the ACCESS command contains entries for only those files that you can reference.

   You should issue an ACCESS command every time you link to a new minidisk with the CP LINK command, to obtain the appropriate file directory.

3. The filename, filetype, and filemode fields can only be specified for disks that are accessed as read-only extensions. For example:

   ```
   access 195 b/a * assemble
   ```

   gives you read-only access to all the files with a filetype of ASSEMBLE on the disk at virtual address 195. The command:

   ```
   access 190 z/a * * z1
   ```

   gives you access to all files on the system disk (190) that have a filemode number of 1.

   When you access any disk in read-only status, files with a filemode number of 0 are not accessed.

4. You can also identify a set of files on a disk by referring to a filename or filetype prefix. For example:

   ```
   access 192 c/a abc*
   ```

   accesses only those files in the disk at virtual address 192 whose filenames begin with the characters ABC. The command line:

   ```
   access 192 c/a * a* c2
   ```

   gives you access to all files whose filetypes begin with an A and which have a filemode number of 2.

5. You can force a read/write disk into read-only status by accessing it as an extension of another disk or of itself; for example:

   ```
   access 191 a/a
   ```

   forces your A-disk into read-only status.

6. When a disk is made a read-only extension of another disk, commands that typically require or allow you to specify a filemode may search extensions of the specified disk. The exceptions to this are the LISTFILE and DISK DUMP commands. For a detailed description of read-only extensions, see the _VM/370 CMS User's Guide_.

7. If you enter the ERASE option by mistake you can recover from the error as long as you have not yet written any new files onto the disk. (That is, you have not yet caused CMS to rewrite the file directory.) Reissue the ACCESS command without the ERASE option.

8. You should never attempt to access a disk in read/write status if another user already has it in read/write status; the results are unpredictable.

9. When accessing OS and DOS disks:

   a. You cannot specify filename, filetype and filemode when you access OS or DOS disks, nor can you specify any options.

   b. In order to see OS and DOS disks, you must have a read/write CMS A-disk available if you are going to use the LOAD command with the MAP option. (MAP is a default option.)

10. If two or more disks have been accessed in CMS, and CP DEFINE commands are executed that swap virtual addresses, then a subsequent RELEASE command may write the file directory on the wrong disk; for example:

```
(CMS) ACCESS 193 C
(CMS) ACCESS 198 E
(CP)  DEFINE 193 293
(CP)  DEFINE 198 193
(CMS) RELEASE C
```

    This sequence of commands will write the file directory from 193 to 198 since the CP definitions are unknown to CMS.

Responses

```
                          ┌     ┐
DMSACC723I mode (cuu) { R/O } |-OS  |
                      { R/W } |-DOS |
                          └     ┘
```

If the specified disk is a CMS disk, this message is displayed if the disk is read-only. If the disk is in OS or DOS format, the message indicates the format, as well as whether it is a read/write or read-only disk.

```
DMSACC724I cuu1 REPLACES mode(cuu2)
```

Before execution of the command, the disk represented by cuu2 was the "mode" disk. The disk, cuu1, is now assigned that filemode letter. This message is followed by message DMSACC726I.

```
                             ┌     ┐
DMSACC725I cuu ALSO = 'mode' |-OS  | DISK
                             |-DOS |
                             └     ┘
```

The disk specified by cuu is the mode disk and an ACCESS command was issued to assign it another filemode letter.

```
DMSACC726I 'cuu mode' RELEASED
```

The disk being accessed at virtual address cuu as a read/write disk is already accessed at a different mode. It is released from that mode. Or, a disk currently accessed at mode is being replaced.

Other Messages and Return Codes

```
DMSACC002E FILE 'DMSROS TEXT' NOT FOUND  RC=28
DMSACC003E INVALID OPTION 'option'  RC=24
DMSACC017E INVALID DEVICE ADDRESS 'cuu'  RC=24
DMSACC048E INVALID MODE 'mode'  RC=24
DMSACC059E 'cuu' ALREADY ACCESSED AS READ/WRITE 'mode' DISK  RC=36
DMSACC060E FILE(S) 'fn [ft  [fm]]' NOT FOUND. DISK  'mode(cuu)' WILL NCT
           BE ACCESSED  RC=28
DMSACC070E INVALID PARAMETER 'parameter'  RC=24
DMSACC109S VIRTUAL STORAGE CAPACITY EXCEEDED  RC=104
DMSACC112S DISK 'mode(cuu)' DEVICE ERROR  RC=100
DMSACC113S mode (cuu) NOT ATTACHED  RC=100
DMSACC230W OS DISK - FILEID AND/OR OPTIONS SPECIFIED ARE IGNORED  RC=4
DMSACC240S ERROR LOADING READ OS ROUTINE 'DMSROS TEXT'
```

AMSERV

Use the AMSERV command to invoke access method services to:

- Define VSAM catalogs, data spaces, or clusters
- Alter, list, copy, delete, export or import VSAM catalogs and data sets

The format of the AMSERV command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│         │                                                                │
│ AMserv  │ fn1   ┌   ┐ [(options...[)]]                                   │
│         │       │fn2│                                                    │
│         │       │fn1│                                                    │
│         │       └   ┘        options:                                    │
│         │                     [PRINT]                                    │
│         │                    ┌              ┐  ┌               ┐         │
│         │                    │TAPIN  { 18n }│  │TAPOUT { 18n  }│         │
│         │                    │       { TAPn}│  │       { TAPn }│         │
│         │                    └              ┘  └               ┘         │
└──────────────────────────────────────────────────────────────────────────┘
```

where:

fn1 specifies the filename of a CMS file with a filetype of AMSERV that contains the access method services control statements to be executed. CMS searches all of your accessed disks, using the standard search order, to locate the file.

fn2 specifies the filename of the CMS file that is to contain the access method services listing; the filetype is always LISTING. If fn2 is not specified, the LISTING file will have the same name as the AMSERV input file (fn1).

The LISTING file is written to the first read/write disk in the standard search order, usually your A-disk. If a LISTING file with the same name already exists, it is replaced.

Options:

PRINT spools the output listing to the virtual printer, instead of writing it to disk. If PRINT is specified, fn2 cannot be specified.

```
TAPIN { 18n  }
      { TAPn }
```

specifies that tape input is on the tape drive at the addresss indicated by 18n or TAPn. n may be 1, 2, 3, or 4, indicating virtual addresses 181 through 184, respectively.

```
TAPOUT { 18n  }
       { TAPn }
```

specifies that tape output should be written to the tape drive at the address indicated by 18n or TAPn. n may be 1, 2, 3, or 4, indicating virtual addresses 181 through 184, respectively.

Note: If both TAPIN and TAPOUT are specified, their virtual device addresses must be different.

Usage Notes

1. To create a job stream for access method services, you can use the CMS Editor to create a file with the filetype of AMSERV. The editor automatically sets input margins at columns 2 and 72.

2. Refer to the *DOS/VS Access Method Services User's Guide* for a description of access method services control statements format and syntax. Restrictions placed on VSAM usage in CMS are listed in this publication in "Appendix B: DOS/VS Access Method Services and VSAM Functions Not Supported in CMS" and "Appendix C: OS/VS Access Method Services and VSAM Functions Not Supported in CMS."

3. You must use the DLBL command to identify the master catalog and all disk input and output files for access method services; the ddname operand of the DLBL command corresponds to the dname parameter following a FILE, INFILE, or OUTFILE keyword in an access method services statement.

4. When you use tape input and/or output with the AMSERV command, you are prompted to enter the ddnames; a maximum of 16 ddnames are allowed for either input and output. The ddnames can each have a maximum of seven characters and must be separated by blanks.

   Since only one tape can be attached at a time for either input or output while using AMSERV, if you you enter more than one tape ddname, the tape files must be in the sequence they are used in the input stream.

5. A CMS format variable file cannot be used directly as input to AMSERV functions as a variable (V) or variable blocked (VB) file because the standard variable CMS record does not contain the BL and RL headers needed by the variable record modules. If these headers are not included in the record, errors will result.

6. If you are using Release 34 of access method services, the "NOLABEL" keyword is available in the environment section of access method services control statements. This keyword is necessary when using AMSERV to read nonlabeled tapes. Tapes created using AMSERV default to nonlabeled tapes.

   All files placed on the CMS disk by AMSERV will show a RECFM of V, even if the true format is fixed (F), fixed blocked (FB), undefined (U), variable or variable blocked. The programmer must know the true format of the file he is trying to use with the AMSERV command and access it properly, or errors will result.

Additional Notes for CMS/DOS Users:

1. You must assign a logical unit to be associated with each ddname named in a DLBL command when you use the AMSERV command in the CMS/DOS environment.

2. AMSERV internally issues an ASSGN command for SYSIPT and locates the source file; therefore, you do not need to assign it. If you use the TAPIN or TAPOUT options, AMSERV also issues ASSGN commands for the tape drives (assigning logical units SYS004 and SYS005).

   Any other assignments and DLBL definitions that are in effect when you invoke the AMSERV command are saved and restored when the command completes executing.

Responses

The CMS ready message indicates that access method services has completed processing. If access method services completed with a nonzero return code, the return code is shown in the ready message. You should examine the LISTING file created by AMSERV to determine the results cf access method services processing.

The publication *DOS/VS Messages* lists and explains all of the messages generated by access method services together with the associated reason codes.

DMSAMS367R ENTER TAPE {INPUT|OUTPUT} DDNAMES:

> This message prompts you to enter the ddnames associated with the tape files.

DMSAMS722I FILE 'fn2 LISTING fm' WILL HOLD AMSERV OUTPUT

> This message is displayed when you enter a fn2 operand or when the listing is not being written on your A-disk; it tells you the file identifier of the output listing.

Other Messages and Return Codes

```
DMSAMS001E NO FILENAME SPECIFIED  RC=24
DMSAMS002E FILE 'fn1 AMSERV' NOT FOUND  RC=28
DMSAMS003E INVALID OPTION 'option'  RC=24
DMSAMS006E NO READ/WRITE DISK ACCESSED FOR 'fn2 LISTING'  RC=36
DMSAMS007E FILE 'fn1 AMSERV fm' NOT FIXED, 80-CHAR. RECORDS  RC=32
DMSAMS065E 'option' OPTION SPECIFIED TWICE  RC=24
DMSAMS066E 'option' AND 'option' ARE CONFLICTING OPTIONS  RC=24
DMSAMS070E INVALID PARAMETER 'parameter'  RC=24
DMSAMS109S VIRTUAL STORAGE CAPACITY EXCEEDED RC=104
DMSAMS113E {TAPIN|TAPOUT} (addr) NOT ATTACHED  RC=100
DMSAMS136S UNABLE TO LOAD 'IDCAMS' RC=104
DMSAMS228E NO DDNAME ENTERED RC=24
DMSSTT062E INVALID  CHARACTER   'char'  IN  FILEID   {'fn1  AMSERV'|'fn2
           LISTING'}  RC=20
```

ASSEMBLE

Use the ASSEMBLE command to invoke the assembler to assemble a file containing source statements. Assembler processing and output is controlled by the options selected. The format of the ASSEMBLE command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
| Assemble | fn     [(options...[)]]                                         |
|          |                                                               |
|          | listing control options:                                      |
|          |                                                               |
|          |      ┌          ┐ ┌      ┐ ┌           ┐ ┌               ┐     |
|          |      |ALOGIC    | |ESD   | |FLAG (nnn) | |LINECOUN (nn) |     |
|          |      |NOALOGIC  | |NOESD | |FLAG (0)   | |LINECOUN (55) |     |
|          |      └          ┘ └      ┘ └           ┘ └               ┘     |
|          |                                                               |
|          |      ┌       ┐ ┌        ┐ ┌          ┐ ┌      ┐ ┌          ┐   |
|          |      |LIST   | |MCALL   | |MLOGIC    | |RLD   | |LIBMAC    |   |
|          |      |NOLIST | |NOMCALL | |NOMLOGIC  | |NORLD | |NOLIBMAC  |   |
|          |      └       ┘ └        ┘ └          ┘ └      ┘ └          ┘   |
|          |                                                               |
|          |      ┌              ┐ ┌        ┐                               |
|          |      |XREF (FULL)   | |PRINT   |                               |
|          |      |XREF (SHORT)  | |NOPRINT |                               |
|          |      |NOXREF        | |DISK    |                               |
|          |      └              ┘ └        ┘                               |
|          |                                                               |
|          | output control options:                                       |
|          |                                                               |
|          |      ┌       ┐ ┌          ┐ ┌        ┐                         |
|          |      |DECK   | |OBJECT    | |TEST    |                         |
|          |      |NODECK | |NOOBJECT  | |NOTEST  |                         |
|          |      └       ┘ └          ┘ └        ┘                         |
|          |                                                               |
|          | SYSTERM options:                                              |
|          |                                                               |
|          |      ┌       ┐ ┌        ┐ ┌          ┐                         |
|          |      |NUMBER | |STMT    | |TERMINAL  |                         |
|          |      |NONUM  | |NOSTMT  | |NOTERM    |                         |
|          |      └       ┘ └        ┘ └          ┘                         |
|          |                                                               |
|          | other assembler options:                                      |
|          |                                                               |
|          |      ┌        ┐ ┌              ┐ ┌        ┐                     |
|          |      |ALIGN   | |BUFSIZE (MIN) | |RENT    |                     |
|          |      |NOALIGN | |BUFSIZE (STD) | |NORENT  |                     |
|          |      └        ┘ └              ┘ └        ┘                     |
|          |      ┌        ┐  ┌                 ┐                           |
|          |      |YFLAG   |  |SYSPARM (string) |                           |
|          |      |NOYFLAG |  |SYSPARM ()       |                           |
|          |      └        ┘  |SYSPARM (?)      |                           |
|          |                 └                 ┘                           |
└──────────────────────────────────────────────────────────────────────────┘
```

where:

fn is the filename of the source file to be assembled and/or the filename of assembler output files. The file must have fixed-length, 80-character records. By default, the assembler expects a CMS file with a filetype of ASSEMBLE.

Listing Control Options: The list below describes the assembler options you can use to control the assembler listing. The default values are underscored.

ALOGIC lists conditional assembly statements in open code.

NOALOGIC suppresses the ALOGIC option.

ESD lists the external symbol dictionary (ESD).

NOESD suppresses the printing of the ESD listing.

FLAG (nnn)
FLAG (0) does not include diagnostic messages and MNOTE messages below severity code nnn in the listing. Diagnostic messages can have severity codes of 4, 8, 12, 16, or 20 (20 is the most severe); and MNOTE message severity codes can be between 0 and 255. For example, FLAG (8) suppresses diagnostic messages with a severity code of 4 and MNOTE messages with severity codes of 0 through 7.

LINECOUN (nn) nn specifies the number of lines to be listed per
LINECOUN (55) page.

LIST produces an assembler listing. Any previous listing is erased.

NOLIST does not produce an assembler listing. However, any previous listing is still erased. This option overrides ESD, RLD, and XREF.

MCALL lists the inner macro instructions encountered during macro generation following their respective outer macro instructions. The assembler assigns statement numbers to these instructions. The MCALL option is implied by the MLOGIC option; NOMCALL has no effect if MLOGIC is specified.

NOMCALL suppresses the MCALL option.

MLOGIC lists all statements of a macro definition processed during macro generation after the macro instruction. The assembler assigns statement numbers to them.

NOMLOGIC suppresses the MLOGIC option.

RLD produces the relocation dictionary (RLD) as part of the listing.

NORLD does not print the relocation directory.

LIBMAC lists the macro definitions read from the macro libraries and any assembler statements following the logical END statement. The logical END statement is the first END statement processed during macro generation. It may appear in a macro or in open code; it may even be created by substitution. The assembler assigns statement numbers to the statements that follow the logical END statement.

NOLIBMAC suppresses the LIBMAC option.

XREF (FULL) includes in the assembler listing a cross-reference table of all symbols used in the assembly. This includes symbols that are defined but never referenced. The assembler listing also contains a cross-reference table of literals used in the assembly.

XREF (SHORT) includes in the assembler listing a cross-reference table of all symbols that are referenced in the assembly. Any symbols defined but not referenced are not included in the table. The assembler listing contains a cross-reference table of literals used in the assembly.

NOXREF does not print the cross-reference tables.

PRINT
PR writes the LISTING file to the printer.

NOPRINT
NOPR suppresses the printing of the LISTING file.

DISK
DI places the LISTING file on a virtual disk.

Output Control Options: The output control options are used to control the object module output of the assembler.

DECK writes the object module on the device specified on the FILEDEF statement for PUNCH. If this option is specified with the OBJECT option, the object module is written both on the PUNCH and TEXT files.

NODECK suppresses the DECK option.

OBJECT
OBJ writes the object module on the device, which is specified by the FILEDEF statement for TEXT, and erases any previous object modules. If this option is specified with the DECK option, the object module is written on the two devices specified in the FILEDEF statement for TEXT and PUNCH.

NOOBJECT
NOOBJ does not create the object module. However, any previous object module is still erased.

TEST includes the special source symbol table (SYM cards) in the object module. This option should not be used for programs to be run under CMS because the SYM cards are not acceptable to the CMS LOAD and INCLUDE commands.

NOTEST Does not produce SYM cards.

SYSTERM Options: The SYSTERM options are used to control the SYSTERM file associated with your assembly.

NUMBER
NUM writes the line number field (columns 73-80 of the input records) in the SYSTERM listing for statements for which diagnostic information is given. This option is valid only if TERMINAL is specified.

NONUM suppresses the NUMBER option.

STMT — writes the statement number assigned by the assembler in the SYSTERM listing for statements for which diagnostic information is given. This option is valid only if TERMINAL is specified.

NOSTMT — suppresses the STMT option.

TERMINAL
TERM — writes the diagnostic information on the SYSTERM data set. The diagnostic information consists of the diagnosed statement followed by the error message issued.

NOTERM — suppresses the TERMINAL option.

Other Assembler Options: The following options allow you to specify various functions and values for the assembler.

ALIGN
ALGN — aligns all data on the proper boundary in the object module; for example, an F-type constant is aligned on a fullword boundary. In addition, the assembler checks storage addresses used in machine instructions for alignment violations.

NOALIGN
NOALGN — does not align data areas other than those specified in CCW instructions. The assembler does not skip bytes to align constants on proper boundaries. Alignment violations in machine instructions are not diagnosed.

BUFSIZE (MIN) — uses the minimum buffer sizes (790 bytes) for each of the utility data sets (SYSUT1, SYSUT2, and SYSUT3). Storage normally used for buffers is allocated to work space. Because more work space is available, more complex programs can be assembled in a given virtual storage size; but the speed of the assembly is substantially reduced.

BUFSIZE (STD) — chooses the buffer size that gives optimum performance. The buffer size depends on the amount of virtual storage. Of the assembler working storage in excess of minimum requirements, 37% is allocated to the utility data set buffers and the rest to macro generation dictionaries.

RENT — checks your program for a possible violation of program reenterability. Code that makes your program nonreenterable is identified by an error message.

NORENT — suppresses the RENT option.

YFLAG — does not suppress the warning messages that indicate that relocatable Y-type address constants have been declared.

NOYFLAG — suppresses the warning messages that indicate relocatable Y-type constants have been declared.

SYSPARM $\left\{\begin{array}{l}\text{(string)}\\ \text{()}\\ \text{(?)}\end{array}\right\}$

passes a character value to the system variable symbol, SYSPARM. The variable (string) cannot be greater than eight characters. If you want to enter a string of more than eight characters, use the SYSPARM (?) format. With the SYSPARM (?) format, CMS prompts you with the message:

```
ENTER SYSPARM:
```

You can enter up to 100 characters. You can also enter parentheses and embedded blanks from the terminal. SYSPARM () enters a null string of characters.

Usage Notes

1. When you issue the ASSEMBLE command, default FILEDEF commands are issued for assembler data sets. You may want to override these with explicit FILEDEF commands. The ddnames used by the assembler are:

ASSEMBLE	(SYSIN input to the assembler)
TEXT	(SYSLIN output of the assembler)
LISTING	(SYSPRINT output of the assembler)
PUNCH	(SYSPUNCH output of the assembler)
CMSLIB	(SYSLIB input to the assembler)
SYSUT1	(workfile of the assembler)
SYSUT2	(workfile of the assembler)
SYSUT3	(workfile of the assembler)

The default FILEDEF commands issued by the assembler for these ddnames are:

```
FILEDEF ASSEMBLE DISK fn ASSEMBLE fm (RECFM FB LRECL 80 BLOCK 800
FILEDEF TEXT DISK fn TEXT fm
FILEDEF LISTING DISK fn LISTING fm (RECFM FBA BLOCK 1210
FILEDEF PUNCH PUNCH
FILEDEF CMSLIB  DISK CMSLIB MACLIB *  (RECFM FB LRECL 80  BLOCK 800
FILEDEF SYSUT1 DISK fn SYSUT1 fm4 (BLOCK 7294 AUXPROC asmproc
FILEDEF SYSUT2 DISK fn SYSUT2 fm4 (BLOCK 7294 AUXPROC asmproc
FILEDEF SYSUT3 DISK fn SYSUT3 fm4 (BLOCK 7294 AUXPROC asmproc
```

At the completion of the ASSEMBLE command, all FILEDEFs that do not have the PERM option are erased.

2. If you want to use any CMS macro or copy libraries during an assembly, you must issue the GLOBAL command to identify the macro libraries before issuing the ASSEMBLE command. For example:

```
global maclib cmslib osmacro testlib
```

identifies the MACLIB files named CMSLIB, OSMACRO, and TESTLIB. If you are invoking ASSEMBLE to assemble code to operate in a VM/370 Basic System Extensions environment, you must add DMSB20 before CMSLIB in the above statement.

3. In order to use OS macro libraries during an assembly, you must issue the FILEDEF command for the OS data set using a ddname of CMSLIB and assigning a CMS file identifier; the filetype must be MACLIB, and you must use the filename on the GLOBAL command line. For example:

```
filedef cmslib disk oldtest maclib c dsn oldtest macros
global maclib oldtest
```

assigns the OS data set OLDTEST.MACROS, on the disk accessed as mode C, a CMS fileid of OLDTEST MACLIB and identifies it as the macro library to be used during assembly.

4. You cannot assemble programs using DOS macros from the DOS/VS source statement libraries under CMS/DOS. You should use the SSERV, ESERV, and MACLIB commands to create CMS MACLIBs to contain DOS macros for assembly under CMS/DOS. See the _VM/370 CMS User's Guide_ for examples.

5. You do not need to make any logical assignments for input or output files when you use the assembler under CMS/DOS. File definitions are assigned by default under CMS, as described in Usage Note 1.

6. Usage information about the VM/370 Assembler Language and assembler options can be found in _OS/VS and VM/370 Assembler Programmer's Guide_ and _OS/VS, DOS/VS, and VM/370 Assembler Language_.

Messages and Return Codes

For the messages and return codes associated with the ASSEMBLE command, see the _OS/VS and VM/370 Assembler Programmer's Guide_.

ASSGN

Use the ASSGN command in CMS/DOS to assign or unassign a system or programmer logical unit for a virtual I/O device. The format of the ASSGN command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
| ASSGN | SYSxxx / Reader   \ [(options...[)]]                               |
|       |       ( PUnch      )                                               |
|       |       ( PRinter    )                                               |
|       |       ( Terminal   )      options:                                 |
|       |       (    ┌ ┐     )                                               |
|       |       < TAP|n|     >     ┌       ┐     ┌      ┐                    |
|       |       (    |1|     (     |UPCASE |     |7TRACK|   [TRTCH a]       |
|       |       (    └ ┘     )     |LOWCASE|     |9TRACK|                   |
|       |       ( mode       )     └       ┘     └      ┘   [DEN den]       |
|       |       ( IGN        )                                               |
|       |        \ UA        /                                               |
└──────────────────────────────────────────────────────────────────────────┘
```

where:

SYSxxx specifies the system or programmer logical unit to be assigned to a particular physical device. SYS000 through SYS241 are valid programmer logical units in CMS/DOS; they may be assigned to any valid device. The system logical units you may assign, and the devices to which they may be assigned, are:

SYSxxx	Valid assignments
SYSRDR	Reader,disk,tape
SYSIPT	Reader,disk,tape
SYSIN	Reader,disk,tape
SYSPCH	Punch,disk,tape
SYSLST	Printer,disk,tape
SYSLOG	Terminal,printer
SYSOUT	Tape
SYSSLB	Disk
SYSRLB	Disk
SYSCLB	Disk
SYSCAT	Disk

The assignment of a system logical unit to a particular device type must be consistent with the device type definition fcr the file in your program.

READER is the spooled card reader (card reader I/O must not be blocked).

PUNCH is the spooled punch.

PRINTER is the spooled printer.

TERMINAL is your terminal (terminal I/O must not be blocked).

TAP[n] is a magnetic tape. n is the symbolic numter of the tape drive. It is either 1, 2, 3, or 4, representing virtual addresses 181, 182, 183, and 184, respectively. If n is omitted, TAP1 is assumed.

mode specifies the one-character mode letter of the disk being assigned to the logical unit (SYSxxx). The disk must be accessed when the ASSGN command is issued.

IGN (ignore) specifies that any attempt to read from the specified device results in an end-of-file indication; any attempt to write to the device is ignored. IGN is not valid when associated with SYSRDR, SYSIPT, SYSIN, or SYSCLB.

UA indicates that the logical unit is to be unassigned. When you release a disk for which an assignment is active, it is automatically unassigned.

Options:

UPCASE translates all terminal input data, to uppercase.

LOWCASE retains all terminal input data as keyed in.

7TRACK
9TRACK is the tape setting.

TRTCH a refers to the tape recording technique for 7-track tapes. Use the following chart to determine the value of a.

a	Parity	Converter	Translator
O	odd	off	off
OC	odd	on	off
OT	odd	off	on
E	even	off	off
ET	even	off	on

DEN den is tape density: den can be 200, 556, 800, 1600, or 6250 bits per inch (bpi). If 200 or 556 are specified, 7TRACK is assumed. If 800, 1600, or 6250 are specified, 9TRACK is assumed. (See Usage Note 8.)

Usage Notes

1. When you enter the CMS/DOS environment with the command SET DOS ON, SYSLOG is assigned by default to TERMINAL. If you specify the mode letter of the DOS/VSE system residence on the SET DOS ON command line, SYSRES is assigned to that disk mode.

2. You cannot assign any of the following DOS/VSE system logical units with the ASSGN command:

```
SYSRES   SYSLNK   SYSVIS
SYSUSE   SYSREC
```

3. If you assign the logical unit SYSIN to a virtual device, SYSRDR and SYSIPT are also assigned to that device. If you make a logical assignment for SYSOUT, both SYSLST and SYSPCH are assigned.

4. To obtain a list of current assignments, use the LISTIO command.

5. To cancel all current assignments (that is, to unassign them), you can enter, in succession, the commands:

```
set dos off
set dos on [mode]
```

6. If you want to access DOS/VSE private libraries, you must assign the logical units SYSSLB (source statement library), SYSRLB (relocatable library), and SYSCLB (core image library), and you must issue the DLBL command to establish a file definition.

7. An assignment to disk (mode) should be accompanied by a DLBL command that provides the disk file identification.

   You cannot make an assignment to a 3350 disk in native mode.

8. If no tape options are specified on the command line, the default for a 7-track tape is 800 bpi, data converter off, translator off and odd parity. If the tape is 9-track, the density defaults to the density of the tape drive. 1600 bpi is the reset condition for 9-track dual-density tapes. If the tape drive is phase-encoded, density defaults to the density of the tape. If the tape drive is NRZI, the reset condition is 800 bpi.

9. 8809 tape drives require the 9TRACK and DEN 1600 options. These are the default options; it is not necessary to state them explicitly.

Responses

None.

Messages and Return Codes

```
DMSASN003E INVALID OPTION 'option'  RC=24
DMSASN027E INVALID DEVICE 'device'  RC=24
DMSASN028E NO LOGICAL UNIT SPECIFIED  RC=24
DMSASN029E INVALID PARAMETER 'parameter' IN THE OPTION 'option'
           FIELD  RC=24
DMSASN035E INVALID TAPE MODE  RC=24
DMSASN050E PARAMETER MISSING AFTER SYSxxx  RC=24
DMSASN065E 'option' OPTION SPECIFIED TWICE  RC=24
DMSASN066E 'option' AND 'option' ARE CONFLICTING OPTIONS  RC=24
DMSASN069E DISK 'mode' NOT ACCESSED  RC=36
DMSASN070E INVALID PARAMETER 'parameter'  RC=24
DMSASN087E INVALID ASSIGNMENT of 'SYSxxx' TO DEVICE 'device'  RC=24
DMSASN090E INVALID DEVICE CLASS 'deviceclass' FOR 'device'  RC=36
DMSASN099E CMS/DOS ENVIRONMENT NOT ACTIVE  RC=40
DMSASN113S '{TAPn|mode|READER|PUNCH|PRINTER} (cuu)' NOT ATTACHED  RC=100
DMSASN366E NO CMS/DOS SUPPORT FOR NATIVE 3350 DISK RC=36
```

CMSBATCH

The system operator uses the CMSBATCH command to invoke the CMS batch facility. Instead of compiling or executing a program interactively, virtual machine users can transfer jobs to the virtual card reader of an active CMS batch virtual machine and thus free up their terminals for other work. The format of the CMSBATCH command is:

```
| CMSBATCH  |  [sysname]                                              |
```

where:

sysname is the eight-character identification of the saved system that is specifically generated for CMS batch operations via the CP SAVESYS command and the NAMESYS macro. Refer to the *VM/370 System Programmer's Guide* for details on SAVESYS and NAMESYS use.

Note: If sysname is not supplied on the command line, then the system that the system operator is currently logged onto becomes the CMS batch virtual machine.

Usage Notes

1. The CMSBATCH command may be invoked immediately after an IPL of the CMS system. Alternatively, BATCH may be specified following the PARM operand on the IPL command line.

2. You should not issue the CMSBATCH command if you use a virtual disk at address 195; the CMS batch virtual machine erases all files on the disk at address 195.

3. For a description of how to send jobs to the CMS batch virtual machine, see the *VM/370 CMS User's Guide*. For an explanation of setting up a batch virtual machine, see the *VM/370 Operator's Guide*.

4. The CMS batch virtual machine can be utilized by personnel who do not have access to a terminal or a virtual machine. This is accomplished by submitting jobs via the real card reader. For details on this, see the *VM/370 CMS User's Guide*.

5. If the CMSBATCH command encounters recursive abends, the message "CMSBATCH system ABEND" appears on the system operator's console.

Error Messages and Return Codes

```
DMSBTB100E NO BATCH PROCESSOR AVAILABLE  RC=40
DMSBTB101E BATCH NOT LOADED  RC= 88
DMSBTP105E NO JOB CARD PROVIDED  RC=None
DMSBTP106E JOB CARD FORMAT INVALID  RC=None
DMSBTP107E CP/CMS COMMAND 'command, (device)' NOT ALLOWED  RC=88
DMSBTP108E /SET CARD FORMAT INVALID  RC=None
DMSBTP109E {CPU|PRINTER|PUNCH} LIMIT EXCEEDED  RC=None
```

COMPARE

Use the COMPARE command to compare two CMS disk files of fixed- or variable-length format on a record-for-record basis and to display dissimilar records at the terminal. The format of the COMPARE command is:

```
┌─────────┬──────────────────────────────────────────────────────────┐
│         │                          ┌     ┐                         │
│ COMpare │ fileid1 fileid2  [(COL mm[-]│nn    │ [)]]                 │
│         │                        1   │lrecl│                       │
│         │                          └     ┘                         │
└─────────┴──────────────────────────────────────────────────────────┘
```

where:

fileid is the file identifier of a file to be compared. All three identifiers (filename, filetype, and filemode) must be specified for each fileid.

Options:

(COL mm-nn)
defines specific columns to be compared. The comparison begins at position mm of each record. The comparison proceeds up to and including column nn. The hyphen (-) may be used in place of a blank if the total number of characters required for mm-nn is not more than eight (maximum parameter field size). If column nn is specified, the hyphen may not follcw or precede a blank. If column nn is not specified, the default ending position is the last character of each record (the logical record length).

Usage Notes

1. To find out whether two files are identical, enter both file identifications, as follows:

   ```
   compare test1 assemble a test1 assemble b
   ```

   Any records that do not match are displayed at the terminal.

2. To stop the display of dissimilar records, use the CMS Immediate command HT.

3. If a file does not exist on a specified disk, that disk's read-only extensions are also searched. The complete fileids of the files being compared are displayed in message DMSCMP179I.

Responses

DMSCMP179I COMPARING 'fn ft fm' WITH 'fn ft fm'

This message identifies the files being compared. If the files are the same (in the columns indicated), this message is followed by the CMS ready message. If any records do not match, the records are displayed. When all dissimilar records have been displayed the message DMSCMP209W is issued.

Other Messages and Return Codes

```
DMSCMP002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSCMP003E INVALID OPTION 'option'  RC=24
DMSCMP005E NO COLUMN SPECIFIED  RC=24
DMSCMP009E COLUMN 'col' EXCEEDS RECORD LENGTH  RC=24
DMSCMP010E PREMATURE EOF ON FILE 'fn ft fm'  RC=40
DMSCMP011E CONFLICTING FILE FORMATS  RC=32
DMSCMP019E IDENTICAL FILEIDS  RC=24
DMSCMP029E INVALID  PARAMETER  'parameter'  IN THE  OPTION  'COL'  FIELD
           RC=24
DMSCMP054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSCMP062E INVALID * IN FILEID    RC=20
DMSCMP104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSCMP209W FILES DO NOT COMPARE  RC=4
DMSCMP211E COLUMN FIELDS OUT OF SEQUENCE  RC=24
```

COPYFILE

Use the COPYFILE command to copy and/or modify CMS disk files. The manner in which the file identifiers are entered determines whether or not one or more output files are created. The format of the COPYFILE command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
| COPYfile | fileidi1 [fileidi2...] [fileido] [(options...[)]]            |
|          |                                                               |
|          |  options:                                                     |
|          |   ┌      ┐ ┌        ┐ ┌        ┐ ┌        ┐                    |
|          |   |Type  | |NEWDate | |NEWFile | |PRompt  |                    |
|          |   |NOType| |OLDDate | |REPlace | |NOPRompt|                    |
|          |   └      ┘ └        ┘ └        ┘ └        ┘                    |
|          |   ┌                 ┐ ┌                  ┐ ┌       ┐            |
|          |   |FRom recno       | |FOR numrec        | |SPecs  |            |
|          |   |FRLabel xxxxxxxx | |TOLabel xxxxxxxx  | |NOSPecs|            |
|          |   └                 ┘ └                  ┘ └       ┘            |
|          |   ┌      ┐ ┌           ┐                     ┌       ┐          |
|          |   |OVly  | |RECfm {F}  |  [LRecl nnnnn]      |TRUnc  |          |
|          |   |APpend| |      {V}  |                     |NOTRunc|          |
|          |   └      ┘ └           ┘                     └       ┘          |
|          |   ┌      ┐ ┌        ┐            ┌        ┐                     |
|          |   |PAck  | |FIll c  |  [EBcdic]  |UPcase  |  [TRans]           |
|          |   |UNPack| |FIll hh |            |LOwcase |                     |
|          |   └      ┘ |FIll 40 |            └        ┘                     |
|          |   [SIngle] └        ┘                                          |
└──────────────────────────────────────────────────────────────────────────┘
```

where:

fileidi1 is the first (or only) input file. Each file identifier (filename, filetype, and filemode) must be specified either by indicating the specific identifier or by coding an asterisk.

fileidi2 is one or more additional input files. Each file identifier (filename, filetype, and filemode) must be specified. In single output mode, any of the three input file identifiers may be specified either by indicating the specific identifier or by coding an asterisk. However, all three file identifiers of fileidi2 cannot be specified by asterisks. In multiple output mode, an asterisk (*) is an invalid file identifier. An equal sign (=) may be coded fcr any of the file identifiers, indicating that it is the same as the corresponding identifier in fileidi1.

fileido is the output file(s) to be created. Each file identifier (filename, filetype, and filemode) must be specified. To create multiple output files, an equal sign (=) must be coded in one or more of the identifier fields. If there is only one input file, fileido may be omitted, in which case it defaults to = = = (the input file represented by fileidi1 is replaced).

The COPYFILE command options are listed below, briefly. For usage notes and examples, see "Using the COPYFILE Command" following the option descriptions.

Options:

TYPE displays, at the terminal, the names of the files being copied.

NOTYPE suppresses the display of the names of the files being copied.

NEWDATE uses the current date as the creation date of the new file(s).

OLDDATE uses the date on the first input file as the creation date of the new file(s).

NEWFILE checks that files with the same fileid as the output file do not already exist. If one or more output files do exist, an error message is displayed and the COPYFILE command terminates. This option is the default so that existing files are not inadvertently destroyed.

REPLACE causes the output file to replace an existing file with the same file identifier. REPLACE is the default option when only one fileid is entered or when the output fileid is specified as "= = =."

PROMPT displays the messages that request specification or translation lists.

NOPROMPT suppresses the display of prompting messages for specification and translation lists.

Copy Extent Options:

FROM recno is the starting record number for each input file in the copy operation.

FRLABEL xxxxxxxx
xxxxxxxx is a character string that appears at the beginning of the first record to be copied from each input file. Up to eight nonblank characters may be specified.

FOR numrec is the number of records to be copied from each input file.

TOLABEL xxxxxxxx
xxxxxxxx is a character string which, if at the beginning of a record, stops the copy operation for that input file. The record containing the given character is not copied. Up to eight nonblank characters may be specified.

SPECS indicates that you are going to enter a specification list to define how records should be copied. See "Entering a COPYFILE Specification List" for information on how you can define output records in a specification list.

NOSPECS indicates that no specification list is to be entered.

OVLY overlays the data in an existing output file with data from the input file. You can use OVLY with the SPECS option to overlay data in particular columns.

APPEND appends the data from the input file at the end of the output file.

Data Modification Options: The following options can be used to change the record format of a file. See "Modifying Record Formats" for more details.

RECFM $\begin{Bmatrix} \text{F} \\ \text{V} \end{Bmatrix}$ is the record format of the output files. If not specified, the output record format is the same as that of the input file.

LRECL nnnnn is the logical record length of the output file(s) if it is to be different from that of the input file(s). The maximum value of nnnnn is 65535.

TRUNC removes trailing blanks (or fill characters) when converting fixed-length files to variable-length format.

NOTRUNC suppresses the removal of trailing blanks (or fill characters) when converting fixed-length files to variable-length format.

PACK compresses records in a file so that they can be stored in packed format.

Caution: A file in packed format should not be modified in any way. If such a file is modified, the UNPACK routines are unable to reconstruct the original file.

UNPACK reverses the PACK operation. If a file is inadvertently packed twice, you can restore the file to its original unpacked form by issuing the COPYFILE command twice.

FILL c
FILL hh
FILL 40

is the padding and truncation character for the TRUNC option or the principal packing character for the PACK option. The fill character may be specified as a single character, c, or by entering a two-digit hexadecimal representation of a character. The default is 40 (the hexadecimal representation for a blank in EBCDIC).

Character Translation Options:

EBCDIC converts a file that was created with 026 keypunch characters (BCD), to 029 keypunch characters (EBCDIC). The following conversions are made:

```
{ to )
& to +
% to (
# to =
@ to '
' to :
```

UPCASE converts all lowercase characters in each record to uppercase before writing the record to the output file.

LOWCASE converts all uppercase characters in each record to lowercase before writing the record to the output file.

TRANS indicates that you are going to enter a list of character translations to be made as the file is copied. See "Entering Translation Specifications" for details on entering a list of characters to be translated.

SINGLE suppresses multiple output mode regardless of the manner in which the file identifiers are specified.

Incompatible Options

Figure 5 shows combinations of options that should not be specified together in the same COPYFILE command. If the option in the first column is specified, none of the options in the second column should be coded.

Option	Incompatible Options
APPEND	LRECL, NEWDATE, NEWFILE, OLDDATE, OVLY, PACK, RECFM, REPLACE, UNPACK
EBCDIC	PACK, UNPACK
FOR	PACK, TOLABEL, UNPACK
FRLABEL	FROM, PACK, UNPACK
FROM	FRLABEL, PACK, UNPACK
LOWCASE	PACK, UNPACK
LRECL	APPEND, PACK, UNPACK
NEWDATE	APPEND, OLDDATE
NEWFILE	APPEND, OVLY, REPLACE
NOPROMPT	PROMPT
NOSPECS	PACK, SPECS, UNPACK
NOTRUNC	PACK, TRUNC, UNPACK
NOTYPE	TYPE
OLDDATE	APPEND, NEWDATE
OVLY	APPEND, NEWFILE, PACK, REPLACE, UNPACK
PACK	APPEND, EBCDIC, FOR, FRLABEL, FROM, LOWCASE, LRECL, OVLY, RECFM, SPECS, TOLABEL, TRANS, TRUNC, UNPACK, UPCASE
PROMPT	NOPROMPT
RECFM	APPEND, PACK, UNPACK
REPLACE	APPEND, NEWFILE, OVLY
SPECS	NOSPECS, PACK, UNPACK
TOLABEL	FOR, PACK, UNPACK
TRANS	PACK, UNPACK
TRUNC	NOTRUNC, PACK, UNPACK
TYPE	NOTYPE
UNPACK	APPEND, EBCDIC, FOR, FRLABEL, FROM, LOWCASE, LRECL, OVLY, PACK, RECFM, SPECS, TOLAEEL, TRANS, TRUNC, UPCASE
UPCASE	PACK, UNPACK

Figure 5. COPYFILE Option Incompatibilities

USING THE COPYFILE COMMAND

The simplest use of the COPYFILE command is for copying a single CMS file from one disk to another, or making a duplicate copy of the file cn the same disk. For example:

```
    copyfile test1 assemble a test2 assemble a
```

makes a copy of the file TEST1 ASSEMBLE A and names it TEST2 ASSEMBLE A.

For those portions of the file identifier that you want to stay the same, you may code an equal sign in the output fileid. Thus, the command line above can be entered:

```
copyfile test1 assemble a test2 = =
```

The equal sign may be used as a prefix or suffix of a file identifier. For example, the command:

```
copyfile a b c file= type= =
```

creates an output file called FILEA TYPEB C.

When you copy a file from one virtual disk to another, you specify the old and new filemodes, and any filename or filetype change you want to make; for example:

```
copyfile test3 assemble c good = a
```

This command makes a copy of the file TEST3 ASSEMBLE C, and names it GOOD ASSEMBLE A.

If you want to copy only particular records in a file, you can use the FROM/FOR FRLABEL/TOLABEL options. For example:

```
copyfile old test a new test a (frlabel start for 41
```

copies 41 records from the file OLD TEST A1, beginning with the record beginning with the character string START into the file NEW TEST A1.

Multiple Input and Output Files

You can combine two or more files into a single file with the COPYFILE command. For example:

```
copyfile test data1 a test data2 = test data3 b
```

copies the files TEST DATA1 and TEST DATA2 from your A-disk and combines them into a file, TEST DATA3, on your B-disk.

Note that if any input file has a filemode number of 3, it is possible that the file will be copied in a sequence different from its order on the disk.

If you want to combine two more files without creating a new file: use the APPEND option. For example:

```
copyfile new list a old list a (append
```

appends the file NEW LIST A to the bottom of the existing file labeled OLD LIST A.

Note: If the file NEW LIST A has a different LRECL from the file OLD LIST A, the appended data is padded, or truncated, to the LRECL of the file OLD LIST A.

Whenever you code an asterisk (*) in an input fileid, you may cause one or more files to be copied, depending upon the number of files that satisfy the remaining conditions. For example:

```
copyfile * test a combined test a
```

copies all files with a filetype of TEST on your A-disk into a single file named COMBINED TEST. If only one file with a filetype of TEST exists, only that file is copied.

If you want to copy all the files on a particular disk to another disk, you could enter:

```
copyfile * * b = = a
```

All the files on the B-disk are copied to the A-disk. The filenames and filetypes remain unchanged.

You can also copy a group of files and change all the filenames or all the filetypes. For example:

```
copyfile * assemble b = test a
```

copies all ASSEMBLE files in the B-disk into files with a filetype of TEST on the A-disk. The filenames are not changed.

You can use the SINGLE option to override multiple output mode. For example:

```
copyfile * test a = = B (single
```

| copies all files on the A-disk with a filetype of TEST to the B-disk as one combined file, with the filename and filetype equal to the first input file found.

Whenever an asterisk appears, it indicates that all files are to be copied; whenever an equal sign (=) appears, it indicates that the same files are to be copied. For example:

```
copyfile x * a1 = file =
```

combines all files with a filename of X on the A-disk into a single file named X FILE A1.

Whenever an equal sign appears in the output fileid in a position corresponding to an asterisk in an input fileid, multiple input files produce multiple output files. When you perform copy operations of this nature you might wish to use the TYPE option, which displays the names of files being copied. For example:

```
copyfile * test a = output a = summary = (type
```

might result in the display:

```
COPY 'ALPHA TEST A1' TO 'ALPHA SUMMARY A1' (NEW FILE)
COPY 'ALPHA OUTPUT A'
COPY 'BETA TEST A1' TO 'BETA SUMMARY A1' (NEW FILE)
COPY 'BETA OUTPUT A.'
```

which indicates that files ALPHA TEST A and ALPHA OUTPUT A were copied into a file named ALPHA SUMMARY A and that files BETA TEST A and BETA OUTPUT A were copied into a file named BETA SUMMARY A.

Modifying Record Formats

You can use the RECFM and LRECL options to change the record format of a file as you copy it. For example:

```
copyfile data file a (recfm f lrecl 130
```

converts the file DATA FILE A1 to fixed-length 130-character records.

If you specify an output fileid, for example:

```
copyfile data file a fixdata file a (recfm f lrecl 130
```

the original file remains unchanged. The file FIXDATA FILE A contains the converted records.

If the records in a file being copied are variable-length, each output record is padded with blanks to the specified record length. If any records are longer than the record length, they are truncated.

When you convert files from fixed-length records to variable-length records, you can specify the TRUNC option to ensure that all trailing blanks are truncated:

```
copyfile data file a (recfm v trunc
```

If you specify the LRECL option and RECFM V, the LRECL option is ignored and the output record length is taken from the longest record in the input file.

When you convert a file from variable-length to fixed-length records, you may also specify a fill character to be used for padding instead of a blank. If you specify:

```
copyfile short recs a (recfm f fill *
```

then each record in the file SHORT RECS is padded with asterisks to the record length. Assuming that SHORT RECS was originally a variable-length file, the record length is taken from the longest existing record. Note that if SHORT RECS is already fixed-length, it is not altered.

Similarly, when you are converting back to variable-length a file that was padded with a character other than a blank, you must specify the FILL option to indicate the pad character, so that character is truncated.

The FILL option can also be used to specify the packing character used with the PACK option. When you use the PACK option, a file is compressed as follows: all occurrences of two or more blanks are encoded as one character, and four or more occurrences of any other character are written as three characters. If you use the FILL option to specify a fill character, then that character is treated as a blank when records are compressed. You must, of course, specify the FILL option to unpack any files packed in this way. Since most fixed-length files are blank-padded to the record length, you do not need to specify the FILL option unless you know that some other character appears more frequently.

A file which is packed on an 800 byte blocksize disk will be fixed format file with a logical record length of 800. On a 1K, 2K, or 4K blocksize disk, the file will be fixed format with a logical record length of 1024. A packed file of either logical record length can be unpacked back to its original specifications regardless of the disk blocksize it resides on. A packed file with logical record length 800 on a disk with blocksize 1K, 2K, or 4K, and packed files with logical record length 1024 on 800 byte disks should be unpacked and re-packed if minimal disk block usage is needed.

When you convert record formats on packed files with the COPYFILE command you can specify single or multiple output files, in accordance with the procedures outlined under "Modifying Record Formats." For example:

```
copyfile * assemble a (pack
```

compresses all ASSEMBLE files in the A-disk without changing any file identifiers. The command:

```
copyfile * assemble a = script = (recfm trunc
```

converts all ASSEMBLE files to variable-length, and changes their filetypes to SCRIPT.

Entering a COPYFILE Specification List

When you use the COPYFILE command, you can specify particular columns of data to be manipulated or particular characters to be translated. Again, how you specify the file identifier determines how many files are copied or modified.

When you use the SPECS option on the COPYFILE command, you receive the message:

```
DMSCPY601R ENTER SPECIFICATION LIST:
```

and a read is presented to your virtual machine and you may enter a specification list. If you do not wish to receive this message, use the NOPROMPT option. The specification list you enter may consist of one or more pairs of operands in the following format:

$$\left\{\begin{array}{l}\text{nn-mm}\\ \text{/string/}\\ \text{hxx...}\end{array}\right\}\ \text{col}$$

where:

nn-mm specifies the start and end columns of the input file that are to be copied to the output file. If mm exceeds the length of the input record, the end of the record is the assumed ending position.

string is any string of uppercase and lowercase characters or numbers delimited by any non-alphameric character.

hxx... is an even number of hexadecimal digits prefixed with an h.

col is the column in the output file at which the copy operation is to begin.

You can enter as many pairs of specifications as you wish. If you want to enter more than one line of specifications, enter two plus signs (++) as continuation indicators.

A specification list may contain any combination of specification pairs; for example:

```
copyfile sorted list a (specs
DMSCPY601R ENTER SPECIFICATION LIST:
/|/ 1 1-8 3 /|/  12 /***/ 14 ++
9-80 18
```

After this command is executed, each record in the file SORTED LIST will look like the following:

```
| oooooooo | *** oooo....
```

where the o's in columns 3 through 10 indicate information originally in columns 1 through 8; the o's following the asterisks indicate the remainder of each record, columns 9 through 80.

When you enter a specification list, you are actually constructing a file column by column. If you specify multiple input or output files, the same copy operation is performed for each record in each output file.

Those columns for which you do not specify any data are filled with blanks or, if you use the FILL option, the fill character of your choice. For example:

```
copyfile sorted list a (specs noprompt lrecl 20 fill $
1-15 6
```

copies columns 1 through 15 beginning in column 6 and writes dollar signs($)in columns 1 through 5.

If you do want to modify data in particular columns of a file but want to leave all of the rest of each record unchanged, you can use the OVLY (overlay) option. For example, the sequence:

```
COPYFILE * bracket a (specs ovly noprompt
had 1 hbd 80
```

overlays the characters [(X'AD') and] (X'BD') in columns 1 and 80 of all the files with a filetype of BRACKET on your A-disk.

When you copy fixed-length files, records are padded or truncated to the record length; variable-length files are always written as specified.

Entering Translation Specifications

You can perform conversion on particular characters in CMS files or groups of files with the TRANS option of the COPYFILE command.

When you enter the TRANS option, you receive the message:

```
DMSCPY602R ENTER TRANSLATION LIST:
```

and a read is presented to your virtual machine. You may enter the translation list. If you do not wish to receive this message, use the NOPROMPT option.

A translation list consists of one or more pairs of characters or hex digits, each pair representing the character you want to translate and the character you want to translate it to, respectively. For example:

```
copy test file a (trans
DMSCPY602R ENTER TRANSLATION LIST:
* - A f0 00 ff
```

specifies that all occurrences of the character * are to be translated to -, all character A's are to be translated to X'F0' and all X'00's are to be translated to X'FF's.

If any translation specifications you enter conflict with the LOWCASE, EBCDIC, or UPCASE options specified on the same command line, the translaticn list takes precedence. In the preceding example, if LOWCASE had also been specified, all A's would be translated to X'F0's, not to a's.

You can enter translation pairs on more than one line if you enter a ++ as a continuation indicator.

Responses

DMSCPY601R ENTER SPECIFICATION LIST:

This message prompts you to enter a specification list when you use the SPECS option.

DMSCPY602R ENTER TRANSLATION LIST:

This message prompts you to enter a translation list when you use the TRANS option.

DMSCPY721I COPY 'fn ft fm' [TO |APPEND| OVLY] 'fn ft fm' [OLD|NEW] FILE

This message appears for each file copied with the TYPE option. It indicates the names of the input file and output file. When you have multiple input files, the output fileid is displayed only once.

Other Messages and Return Codes

```
DMSCPY002E {INPUT|OVERLAY} FILE 'fn ft fm' NOT FOUND RC=28
DMSCPY003E INVALID OPTION 'option'  RC=24
DMSCPY024E FILE 'fn ft fm' ALREADY EXISTS -- SPECIFY 'REPLACE'  RC=28
DMSCPY029E INVALID PARAMETER  'parameter' IN  THE OPTION  'option' FIELD
           RC=24
DMSCPY030E FILE 'fn ft fm' ALREADY ACTIVE  RC=28
DMSCPY037E DISK 'mode' IS READ/ONLY  RC=36
DMSCPY042E NO FILEID(S) SPECIFIED  RC=24
DMSCPY048E INVALID MODE 'mode'  RC=24
DMSCPY054E INCOMPLETE FILEID 'fn [ft'] SPECIFIED  RC=24
DMSCPY062E INVALID CHAR '[=|*|char]' IN FILEID '[fn ft fm]'  RC=20
DMSCPY063E NO {TRANSLATION|SPECIFICATION} LIST ENTERED  RC=40
DMSCPY064E INVALID  [TRANSLATE]  SPECIFICATION  AT  OR  NEAR  '........'
           RC=24
DMSCPY065E 'option' OPTION SPECIFIED TWICE  RC=24
DMSCPY066E 'option' AND 'option' ARE CONFLICTING OPTIONS  RC=24
DMSCPY067E COMBINED  INPUT FILES  ILLEGAL WITH  PACK  OR UNPACK  OPTIONS
           RC=24
DMSCPY068E INPUT FILE 'fn ft fm' NOT IN PACKED FORMAT  RC=32
DMSCPY101S 'SPECS' TEMP STRING STORAGE EXHAUSTED AT '........'  RC=88
DMSCPY102S TOO MANY FILEIDS  RC=88
DMSCPY103S NUMBER OF SPECS EXCEEDS MAX 20  RC=88
DMSCPY156E 'FROM nnn' NOT FOUND --FILE 'fn ft fm' HAS ONLY 'nnn' RECORDS
           RC=32
DMSCPY157E LABEL 'label' NOT FOUND IN FILE 'fn ft fm'  RC=32
DMSCPY172E TO LABEL 'label' {EQUALS| IS AN INITIAL SUBSTRING OF} FRLABEL
           'label' RC=24
DMSCPY173E NO RECORDS WERE COPIED TO OUTPUT FILE 'fn ft fm'  RC=40
DMSCPY901T UNEXPECTED ERROR  AT 'addr':  PLIST 'plist'  AT 'addr',  BASE
           'addr', RC 'nn'  RC=256
DMSCPY903T IMPOSSIBLE PHASE CODE 'xx'  RC=256
DMSCPY904T UNEXPECTED UNPACK ERROR AT 'addr', BASE 'addr' RC=256
```

CP

Use the CP command to transmit commands to the VM/370 control program environment without leaving the CMS environment. The format of the CP command is:

```
| CP    | [ commandline ]                                          |
```

where:

commandline is any CP command valid for your CP command privilege class. If this field is omitted, you are placed in the CP environment and may enter CP commands without preceding each command with CP. To return to CMS, issue the CP command BEGIN.

Usage Notes

1. You must use the CP command to invoke a CP command:
    - Within an EXEC procedure
    - If the implied CP (IMPCP) function is set to OFF for your virtual machine
    - In a job you send to the CMS batch facility
2. To enter a CP command from the CMS environment without CMS processing the command line, use the #CP function.
3. When you enter an invalid CP command following the CP command, you receive a return code of -1. In an EXEC, this return code is +1.

Responses

All responses are from the CP command that was issued, and are followed by the CMS ready message.

DDR

Use the DASD Dump Restore (DDR) program to dump, restore, copy, or print VM/370 user minidisks. The DDR program may run as a standalone program, or under CMS via the DDR command.

The DDR program has five functions:

1. Dumps part or all of the data from a DASD device to tape.
2. Transfers data from tapes created by the DDR dump function to a direct access device. The direct access device must be the same as that which originally contained the data.
3. Copies data from one device to another of the same type. Data may be reordered, by cylinder or by block for fixed-block DASD, when copied from disk to disk. In order to copy one tape to another, the original tape must have been created by the DDR DUMP function.
4. Prints selected parts of DASD and tape records in hexadecimal and EBCDIC on the virtual printer.
5. Displays selected parts of DASD and tape records in hexadecimal and EBCDIC on the terminal.

The format of the DDR command is:

```
┌──────────────────────────────────────────────────────────────────────┐
|                |               ┌  ┐                                  |
|   DDR          | [fn   ft |fm|  ]                                     |
|                |               |* |                                  |
|                |               └  ┘                                  |
└──────────────────────────────────────────────────────────────────────┘
```

where:

```
       ┌  ┐
fn ft |fm|  is the identification of the file containing the control
      |* |  statements for the DDR program. If no file
       └  ┘  identification is provided, the DDR program attempts to
             obtain control statements from the console. The filemode
             defaults to * if a value is not provided.
```

Note: If you use the CMS DDR command, CMS ignores the SYSPRINT control statement and directs the output to the CMS printer 00E.

Note: Be aware that DDR when run as a standalone program does not have error recovery support. However, when DDR is invoked in CMS, in a virtual machine environment, the I/O operation is performed by CP (CP has better error recovery facilities).

DDR CONTROL STATEMENTS

DDR control statements describe the intended processing and the needed I/O devices. I/O definition statements must be specified first.

All control statements may be entered from either the console or the card reader. Only columns 1 to 71 are inspected by the program. All data after the last operand in a statement is ignored. An output tape must have the DASD cylinder header records in ascending sequences; therefore, the extents must be entered in sequence by DASD location, that is, in sequence by cylinder number if count-key-data or by block number if FB-512. Only one type of function -- dump, restore, or copy -- may be performed in one execution, but up to 20 statements describing cylinder or block extents may be entered. The function statements are delimited by an INPUT or OUTPUT statement, or by a null line if the console is used for input. If additional functions are to be performed, the sequence of control cards must be repeated. If you do not use INPUT or OUTPUT control statements to separate the functions you specify when the input is read from a card reader or CMS file, an error message (DMKDDR702E) is displayed. The remainder of the input stream will be checked for proper syntax, but no further DDR operations will be performed. Only those statements needed to redefine the I/O devices are necessary for subsequent steps. All other I/O definition remain the same.

To return to CMS, enter a null line (carriage return) in response to the prompting message (ENTER:). To return directly to CP, key in #CP.

The PRINT and TYPE statements work differently from other DDR control statements in that they operate on only one data extent at a time. If the input is from a tape created by the dump function, it must be positioned at the header record for each step. The PRINT and TYPE statements have an implied output of either the console (TYPE) or system printer (PRINT). Therefore, PRINT and TYPE statements need not be delimited by an INPUT or OUTPUT statement.

I/O DEFINITION STATEMENTS

The I/O definition statements describe the tape, DASD, and printer devices used while executing the DASD Dump Restore program.

INPUT/OUTPUT Control Statement

An INPUT or OUTPUT statement describes each tape and DASD unit used. The format of the INPUT/OUTPUT statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
│            │                ┌       ┐                                │
│  INput     │   cuu   type   │volser│  [(options...)]                 │
│  OUTput    │                │altape│                                 │
│            │                └       ┘                                │
│            │        Options:                                         │
│            │        ┌         ┐ ┌             ┐ ┌        ┐            │
│            │        │SKip  nn │ │MOde   6250 │ │REWind│              │
│            │        │SKip  0  │ │MOde   1600 │ │UNload│              │
│            │        └         ┘ │MOde    800 │ │LEave │              │
│            │                    └             ┘ └        ┘            │
└──────────────────────────────────────────────────────────────────────┘
```

where:

INPUT indicates that the device described is an input device.

OUTPUT indicates that the device described is an output device.

Note: If the output device is a DASD device and DDR is running under CMS, the device is released using the CMS RELEASE command function and DDR processing continues.

cuu — is tne unit address of the device.

type — is the device type (2314, 2319, 3330, 3330-11, 3340-35, 3340-70, 3350, 2305-1, 2305-2, FB-512(FB), 2400, 2420, 3420, or 8809) (no 7-track support for any tape devices). Specify a 3410 as a 3420. Specify a 3340-70F as a 3340-70, and a 3333 as a 3330. Specify a 3350 that is in 3330-1 or 3330-11 compatibility mode as a 3330 or 3330-11. Specify a 3344 as a 3340-70, and specify 3350 for a 3350 operating in native mode (as opposed to compatibility mode). Note that both 3310 and 3370 are denoted by specifying FB-512 or simply FB.

Note: The DASD Dump Restore (DDR) program, executing in a virtual machine, uses I/O DIAGNOSE 20 to perform I/O operations on tape and DASD devices. DDR under CMS requires that the device type entered agree with the device type of the real device as recognized by VM/370. If there is a conflict with device types, the following message is issued:

```
DMKDDR708E INVALID OPTION
```

However, if DDR executes standalone in a virtual machine, DDR uses DIAGNOSE 20 to perform the I/O cperation if the device types agree. If the device types do not agree, error message DMKDDR708E is issued.

The speed setting for 8809 tape drives is not under the user's control. When DDR is running as a command under CMS, the 8809 is supported only in start/stop mode. If DDR is run stand-alone in a virtual machine, DDR attempts to run the 8809 in high-speed mode. In this mode, the data transfer time is reduced. However, this does not mean that the time for a DDR job is reduced; job duration depends on many factors such as processor and device contention.

volser — is the volume serial number of a DASD device. If the keyword "SCRATCH" is specified instead of the volume serial number, no label verification is performed.

altape — is the address of an alternate tape drive.

Note: If multiple reels of tape are required and "altape" is not specified, DDR types the following at the end of the reel:

```
END OF VOLUME CYL xxx HD xxx, MOUNT NEXT TAPE
```

After the new tape is mounted, DDR continues automatically.

Options:

```
SKIP nn
     0
```

forward spaces nn files on the tape. nn is any number up to 255. The SKIP option is reset to zero after the tape has been positioned.

```
     ┌    ┐
MODE │6250│
     │1600│
     │ 800│
     └    ┘
```

causes all output tapes that are opened for the first time and at the load point to be written or read in the specified density. All subsequent tapes mounted are also set to the specified density. If no mode option is specified, then no mode set is performed and the density setting remains as it previously was.

REWIND — rewinds the tape at the end of a function.

UNLOAD — rewinds and unloads the tape at the end of a function.

LEAVE leaves the tape positioned at the end of the file at the end of a function.

Notes:

1. When the wrong input tape is mounted, the message DMKDDR709E is displayed and the tape will rewind and unload regardless of options REWIND, UNLOAD, or LEAVE being specified.

2. If DDR is executed from CMS, failure to attach the tape drive or the disk device (or both) to your virtual machine prior to invoking the input/output statement causes the following response to be displayed:

   INVALID INPUT OR OUTPUT DEFINITION

SYSPRINT Control Statement

Use the SYSPRINT control statement (in the standalone DDR virtual machine only) to describe the printer that is to print data extents specified by the PRINT statement. It also can print a map of the cylinder extents from the DUMP, RESTORE, or COPY statement. If the SYSPRINT statement is not provided, the printer assignment defaults to 00E. CMS ignores the SYSPRINT statement when you invoke DDR as a command under CMS, and CMS always directs the output to 00E. The format of the SYSPRINT control statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
│ SYsprint │  cuu                                                      │
└──────────────────────────────────────────────────────────────────────┘
```

where:

cuu   specifies the unit address of the device.

Function Statements

The function statements tell the DDR program what action to perform. The function commands also describe the extents to be dumped, copied, or restored. The format of the DUMP/COPY/RESTORE control statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
│           │ ┌                                                ┐       │
│   DUmp    │ │cyl1 [To]  [cyl2 [Reorder] [To] [cyl3]]          │       │
│   COpy    │ │block1 [To] [block2 [Reorder] [To] [block3] ]    │       │
│   REstore │ │CPvol                                            │       │
│           │ │ALL                                              │       │
│           │ │NUcleus                                          │       │
│           │ └                                                ┘       │
└──────────────────────────────────────────────────────────────────────┘
```

where:

DUMP   requests the program to move data from a direct access volume onto a magnetic tape or tapes. The format of the tape depends on the type of the direct access volume. The tape format is shown for both count-key-data and FB-512 devices.

For count-key-data DASD, the data is moved cylinder-by-cylinder. Any number of cylinders can be moved. The format of the resulting tape is:

Record 1: a volume header record, consisting of data describing the volumes.

Record 2: a track header record, consisting of a list of count fields to restore the track, and the number of data records written on tape. After the last count field the record contains key and data records to fill the 4K buffer.

Record 3: track data records, consisting of key and data records packed into 4K blocks, with the last record truncated.

Record 4: either the end-of-volume (EOV) or end-of-job (EOJ) trailer label. The end-of-volume label contains the same information as the next volume header record, except that the ID field contains EOV. The end-of-job trailer label contains the same information as record 1 except that the cylinder number field contains the disk address of the last record on tape and the ID field contains EOJ.

For FB-512 devices, the data is moved in 'sets' of blocks. Each set contains 95 blocks of data. (The last set moved may have less than 95 blocks.) Any number of blocks can be moved with one DUMP statement. The format of the resulting tape is:

Record 1: a volume header record, consisting of data describing the volume.

Record 2: a data header record. This consists of control data that describes the set of blocks that follow (such as block numbers and the number of 4K tape records required to hold these FB-512 blocks). Following the control data is the actual FB-512 blocks filling out the 4K tape record.

Record 3: FB-512 data records. These contain the rest of the blocks making up the set.

Record 4: either the end-of-volume (EOV) or end-of-job (EOJ) trailer label. The EOV label contains the same information as the next header record, except that the ID field contains EOV. The EOJ trailer label is just like record 1 except that it contains the number of the last DASD block dumped and the ID field contains EOJ.

COPY requests the program to copy data from one device to another device of the same or equivalent type. Note that you cannot copy between FB-512 and count-key-data DASD. Data may be recorded on a cylinder or block basis from input device to output device. A tape-to-tape copy can be accomplished only with data dumped by this program.

RESTORE requests the program to return data that has been dumped by this program. Data can be restored only to a DASD volume of the same or equivalent device type from which it was dumped. It is possible to dump from a real disk and restore to a minidisk as long as the device types are the same.

cyl1 [TO] [cyl2 [REORDER] [TO] [cyl3]]

Only those cylinders specified are moved, starting with the first track of the first cylinder (cyl1), and ending with the last track of the second cylinder (cyl2). The REORDER operand causes the output to be reordered, that is, moved to different cylinders, starting at the specified cylinder (cyl3) or at the starting cylinder (cyl1) if cyl3 is not specified. The REORDER operand must not be specified unless specified limits are defined for the operation; the starting and, if required, ending cylinders (cyl1 and cyl2) must be specified. Note that if the input device cylinder extents exceed the number of cylinders specified on the output device, an error message results.

block1 [To] [block2 [Reorder] [To] [block3]]

Only those blocks specified are moved, starting with the block indicated by block1, up to and including the block indicated by block2. The REORDER operand causes the data to be moved to a different DASD location. The REORDER operand must not be specified unless specified limits are defined for the operation. If the input block extents exceed the capacity of the output device, an error message results.

CPVOL specifies that cylinder 0 (blocks 0-15 if FB-512) and all active directory and permanent disk space are to be copied, dumped, or restored. This indicates that both source and target disk must be in CP format; that is, the CP Format/Allocate program must have formatted them.

ALL specifies that the operation is to be performed on the entire DASD volume (all cylinders or all blocks).

Note: The occurrence of message DMKDDR705E (issued upon completion of the copy, restore, or dump operation) indicates that an attempt was made to copy, restore, or dump the contents of DASD locations beyond the extents of the designated minidisk.

NUCLEUS specifies that record 2 on cylinder 0, track 0 and the nucleus on cylinder 0, track 0 (blocks 5-12 if FB-512) are dumped, copied, or restored.

Restrictions (for other than FB-512):

- Each track must contain a valid home address, containing the real cylinder and track location.
- Record zero must not contain more than eight key and/or data characters.
- Flagged tracks are treated just as any other track for all 2314, 2319, 3340, and 2305 devices. That is, no attempt is made to substitute the alternate track data when a defective primary track is read. In addition, tracks are not inspected to determine whether they were previously flagged when written. Therefore, volumes

containing flagged tracks should be restored to the same cylinders of the volume from which they were dumped. The message DMKDDR715E occurs each time a defective track is dumped, copied or restored, and the operation continues.

- Flagged tracks on 3330, and 3350 devices are handled automatically by the control unit and may never be detected by the program. The program may detect a flagged track if, for example, no alternate track is assigned to the defective primary track. If a flagged track is detected by the program, the message DMKDDR715E occurs and the operation terminates.

Example:

```
INPUT 191 3330 SYSRES
OUTPUT 180 2400 181 (MODE 800
SYSPRINT 00F
DUMP CPVOL
INPUT 130 3330 MINI01
DUMP 1 TO 50 REORDER 51
60 70 101
```

This example sets the density to 800 bpi, then dumps all pertinent data from the volume labeled SYSRES onto the tape that is mounted on unit 180. If the program runs out of space on the first tape, it continues dumping onto the alternate device (181). A map of the dumped cylinders is printed on unit 00F while the program is dumping. When the first function is complete, the volume labeled MINI01 is dumped onto a new tape. Its cylinder header records are labeled 51 to 100. A map of the dumped cylinders is printed on unit 00F. Next, cylinders 60 to 70 are dumped and labeled 101 to 111. This extent is added to the cylinder map on unit 00F. When the DDR processing is complete, the tapes are unloaded and the program stops.

If cylinder extents are being defined from the console, the user need only enter DUMP, COPY or RESTORE on the command line. The following is displayed:

```
ENTER CYLINDER EXTENTS
ENTER:
```

For any extent after the first extent, the message:

```
ENTER NEXT EXTENT OR NULL LINE
ENTER:
```

is displayed.

You may then enter additional extents to be dumped, restored, or copied. A null line causes the job step to start.

Notes:

1. When a cylinder map is printed on the virtual printer (00F as in the previous example) a heading precedes the map information. Module DMKDDR controls the disk, time and zone printed in the heading. Your installation must apply a local modification to DMKDDR to ensure that local time, rather than GMT (Greenwich Meridian Time), is printed in the heading.

| 2. Attempts to restore cylinders or blocks beyond the capacity that had been recorded on the tape produces a successful EOJ, but the | printout only indicates the last cylinder or block found on the tape.

PRINT/TYPE Function Statement

Use the PRINT and TYPE function statement to print or type (display) a hexadecimal and EBCDIC translation of each record specified. The input device must be defined as direct access tape. The output is directed to the system console for the TYPE function, or to the SYSPRINT device for the PRINT function. (This does not cause redefinition of the output unit definition.) The format of the PRINT/TYPE control statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
| PRint    |[cyl1 [hh1 [rr1]] [To cyl2 [hh2 [rr2 ]]] [ (options...[) ]]]|
| TYpe     |[block1 [To block2]                                        ]|
|          |          options:                                          |
|          |          [Hex] [Graphic] [Count]                           |
└──────────────────────────────────────────────────────────────────────┘
```

where:

cyl1 is the starting cylinder.

hh1 is the starting track. If present, it must follow the cyl1 operand. The default is track zero.

rr1 is the starting record. If present, it must follow the hh1 operand. The default is home address and record zero.

TO cyl2 is the ending cylinder. If more than one cylinder is to be printed or typed, "TO cyl2" must be specified.

hh2 is the ending track. If present, it must follow the cyl2 operand. The default is the last track on the ending cylinder.

rr2 is the record ID of the last record to print. The default is the last record on the ending track.

block1 is the starting FB-512 block number.

To block2 is the ending block number. If more than one block is to be printed or typed, 'To block2' must be specified.

Options:

HEX prints or displays a hexadecimal representation of each record specified.

GRAPHIC prints or displays an EBCDIC translation of each record specified.

COUNT prints or displays only the count field for each record specified. This option is ignored for FB-512 data.

Usage

If the TYPE statement follows the occurrence of error message DMKDDR705E and specifies the same cylinder, track, and record extents indicated in the error message, the contents of the printed record must be interpreted in the context of the I/O error information given in the initial message.

Examples

PRINT 0 TO 3

| Prints all of the records from cylinders or blocks 0, 1, 2, and 3.

PRINT 0 1 3

Prints only one record, from cylinder 0, track 1, record 3.

PRINT 1 10 3 TO 1 15 4

Prints all records starting with cylinder 1, track 10, record 3, and ending with cylinder 1, track 15, record 4.

The example in Figure 6 shows the information displayed at the console (TYPE function) or system printer (PRINT function) by the DDR program. The listing is annotated to describe some of the data fields.

| The printed output for FB-512 data is self-explanatory. DDR prints a short heading telling the block number, then prints the 512 bytes of data in that block.

Responses

DMKDDR711R VOLID READ IS volid2 [NOT volid1]
DO YOU WISH TO CONTINUE? RESPOND YES NO OR REREAD:

where:

volid2 is the volume serial number from the VOL1 label on the DASD unit.

volid1 is the volume serial number from the INPUT or OUTPUT control card.

The volume serial number read from the device at cuu is not the same as that specified on the INPUT or OUTPUT control card.

DMKDDR716R NO VOL1 LABEL FOUND FOR volser
DO YOU WISH TO CONTINUE? RESPOND YES NO OR REREAD:

where:

volser is the volume serial number of the DASD device from the INPUT or the OUTPUT control card.

The DASD device at cuu contains no volume serial number.

DMKDDR717R DATA DUMPED FROM volid1 TO BE RESTORED TO volid2
DO YOU WISH TO CONTINUE? RESPOND YES NO OR REREAD:

where:

volid1 is the volume serial number from the input tape header record (volume dumped).

volid2 is the volume serial number from the output DASD device.

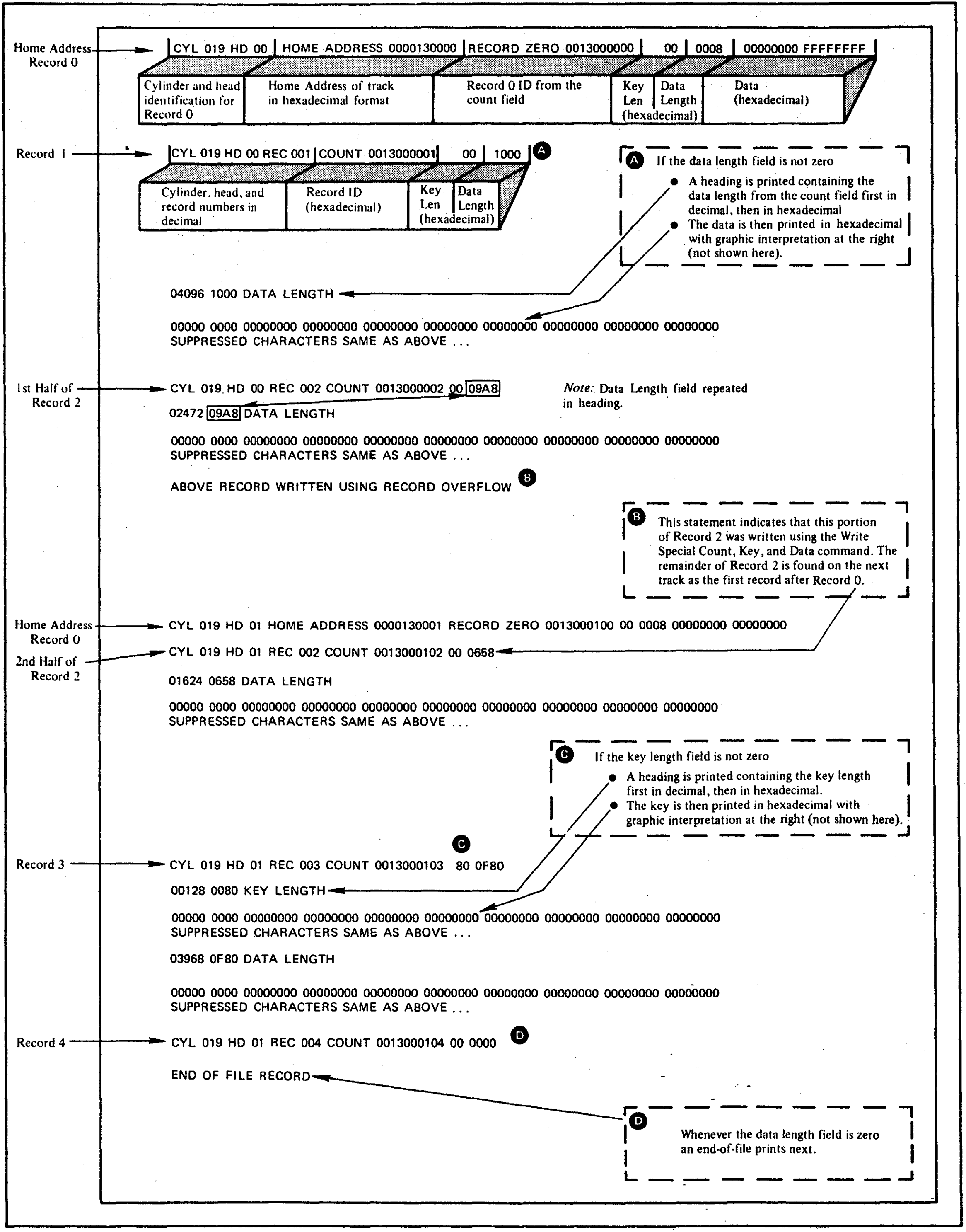
The above message is printed to verify the input parameters.

| ENTER CYLINDER EXTENTS or ENTER BLOCK EXTENTS
| ENTER: ENTER:

This message is received only if you are entering input from your terminal.

| END OF VOLUME CYL xxx HD xx, MOUNT NEXT TAPE
or END OF BLOCK xxxxxxxx, MOUNT NEXT TAPE

DDR continues processing, after the mounting of the next tape reel.



```
CYL 019 HD 00 | HOME ADDRESS 0000130000 |RECORD ZERO 0013000000 |  00 | 0008 | 00000000 FFFFFFFF

CYL 019 HD 00 REC 001 |COUNT 0013000001|  00 | 1000

04096 1000 DATA LENGTH

00000 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
SUPPRESSED CHARACTERS SAME AS ABOVE ...

CYL 019 HD 00 REC 002 COUNT 0013000002 00 09A8
02472 09A8 DATA LENGTH

00000 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
SUPPRESSED CHARACTERS SAME AS ABOVE ...

ABOVE RECORD WRITTEN USING RECORD OVERFLOW

CYL 019 HD 01 HOME ADDRESS 0000130001 RECORD ZERO 0013000100 00 0008 00000000 00000000
CYL 019 HD 01 REC 002 COUNT 0013000102 00 0658

01624 0658 DATA LENGTH

00000 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
SUPPRESSED CHARACTERS SAME AS ABOVE ...

CYL 019 HD 01 REC 003 COUNT 0013000103  80 0F80

00128 0080 KEY LENGTH

00000 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
SUPPRESSED CHARACTERS SAME AS ABOVE ...

03968 0F80 DATA LENGTH

00000 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
SUPPRESSED CHARACTERS SAME AS ABOVE ...

CYL 019 HD 01 REC 004 COUNT 0013000104 00 0000

END OF FILE RECORD
```

A If the data length field is not zero

- A heading is printed containing the data length from the count field first in decimal, then in hexadecimal
- The data is then printed in hexadecimal with graphic interpretation at the right (not shown here).

Note: Data Length field repeated in heading.

B This statement indicates that this portion of Record 2 was written using the Write Special Count, Key, and Data command. The remainder of Record 2 is found on the next track as the first record after Record 0.

C If the key length field is not zero

- A heading is printed containing the key length first in decimal, then in hexadecimal.
- The key is then printed in hexadecimal with graphic interpretation at the right (not shown here).

D Whenever the data length field is zero an end-of-file prints next.

Figure 6. An Annotated Sample of Output from the TYPE and PRINT Functions of the DDR Program

RESTORING volser

where:

volser is the volume serial number of the disk dumped.

The RESTORE operation has begun.

COPYING volser

where:

volser is the volume serial number described by the input unit.

The COPY operation has begun.

DUMPING volser

where:

volser is the volume serial number described by the input unit.

The DUMP operation has begun.

PRINTING volser

where:

volser is the volume serial number described by the input unit.

The PRINT operation has begun.

END OF DUMP

The DUMP operation has ended.

END OF RESTORE

The RESTORE operation has ended.

END OF COPY

The COPY operation has ended.

END OF PRINT

The PRINT operation has ended.

END OF JOB

All specified operations have completed.

ENTER:

Prompts for input from the terminal. A null line (that is, pressing the Enter key or equivalent) causes control to return to CMS if the virtual machine is in the CMS environment.

DMKDDR725R ORIGINAL INPUT DEVICE WAS(IS) LARGER THAN OUTPUT DEVICE. DO YOU WISH TO CONTINUE? RESPONSE YES OR NO:

Explanation:
RESTORE function - The number of cylinders or blocks on the original DASD input unit is compared with the number on the output device.

COPY function - The input device contains more cylinders or blocks than the output device.

Operator Action: The operator must determine if the COPY or RESTORE function is to continue. The response is either yes or no.

Other Messages and Return Codes

Note: Except as shown, there is no return code returned for the following messages. For FB-512 devices, DASD locations are described by a specific block number instead of by cchhr.

```
DMKDDR700E INPUT UNIT IS NOT A CPVOL
DMKDDR701E INVALID OPERAND - operand
DMKDDR702E CONTROL STATEMENT SEQUENCE ERROR
DMKDDR703E OPERAND MISSING
DMKDDR704E DEV cuu NOT OPERATIONAL
DMKDDR705E IO ERROR  cuu CSW csw  SENSE sense INPUT  bbcchh|block OUTPUT
           bbcchh|block CCW ccw
DMKDDR707E MACHINE CHECK RUN SEREP AND SAVE OUTPUT FOR CE
DMKDDR708E INVALID INPUT OR OUTPUT DEFINITION
DMKDDR709E WRONG INPUT TAPE MOUNTED
DMKDDR710A DEV cuu INTERVENTION REQUIRED
DMKDDR712E NUMBER OF EXTENTS EXCEEDS 20
DMKDDR713E OVERLAPPING OR INVALID EXTENTS
DMKDDR714E RECORD bbcchh|block NOT FOUND ON TAPE
DMKDDR715E LOCATION bbcchh|block IS A FLAGGED TRACK  RC=3
DMKDDR718E OUTPUT UNIT IS FILE PROTECTED  RC=1
DMKDDR719E INVALID FILENAME OR FILE NOT FOUND
DMKDDR720E ERROR IN routine  RC=varies
DMKDDR721E RECORD cchhr|block NOT FOUND
DMKDDR722E OUTPUT UNIT NOT PROPERLY FORMATTED FOR THE CP NUCLEUS
DMKDDR723E NO VALID CP NUCLEUS ON THE INPUT UNIT
DMKDDR724E INPUT TAPE CONTAINS A CP NUCLEUS DUMP
DMKDDR756E PROGRAM CHECK PSW=psw
```

DEBUG

Use the DEBUG command to enter the debug environment from the CMS environment. In the debug environment you can use a variety of DEBUG subcommands that allow you to test and debug your programs. The DEBUG subcommands are described in "Section 4. DEBUG Subcommands." For tutorial information, including examples, see the *VM/370 CMS User's Guide*. The format of the DEBUG command is:

```
| DEBUG |
```

Usage Notes

1. The debug environment is also entered as a result of an external interruption or the result of a breakpoint (address stop) encountered during program execution.

2. Once you are in the debug environment, you can enter only DEBUG subcommands and CP commands via the #CP function.

3. To return to the CMS environment, enter the DEBUG subcommand RETURN.

Responses

DMSDBG728I DEBUG ENTERED

This message indicates that you are in the debug environment.

DISK

Use the DISK command to:

- Punch CMS disk files to the virtual spooled card punch in a special format which allows the punched deck to be restored to disk in the form of the original disk file.

- Restore punched decks created by the DISK DUMP command to a disk file.

The format of the DISK command is:

```
| DISK | {DUMP  fn ft [fm]  } |
|      | {LOAD              } |
```

where:

DUMP fn ft fm
punches the specified file (fn ft fm). The file may have either fixed- or variable-length records. After all data is punched, an end-of-file card is created with an N in column 5. This card contains directory information and must remain in the deck. The original disk file is retained.

LOAD
loads a file or files from the spooled card reader and writes them as CMS files on your A-disk. The filename and filetype are obtained from the card stream. If a file exists with the same filename and filetype as one of those in the card stream, it is replaced.

Note: DISK LOAD file identifiers are those of the specified file issued by the DISK DUMP command.

Usage Notes

1. To read files with the DISK LOAD command, they must have been created by the DISK DUMP command. To load spooled reader files created in any other manner, you should use the READCARD command.

2. To load reader files created by DISK DUMP, you must issue the DISK LOAD command for each spool file. For example, if you enter:

```
disk dump source1 assemble
disk dump source2 assemble
```

the virtual machine that receives the files must issue the DISK LOAD command twice to read the files onto disk. If you use the CP SPOOL command to spool continuous, for example:

```
cp spool punch cont
disk dump source1 assemble
disk dump source2 assemble
cp spool punch nocont close
```

then you only need to issue the DISK LOAD command once to read both files.

Responses

There is no response to the DISK DUMP command. The file identifiers of each file loaded are displayed when you issue the DISK LOAD command:

```
fn ft fm
.  .  .
.  .  .
.  .  .
```

Other Messages and Return Codes

```
DMSDSK002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSDSK014E INVALID FUNCTION 'function'  RC=24
DMSDSK037E DISK 'A' IS READ/ONLY  RC=36
DMSDSK047E NO FUNCTION SPECIFIED  RC=24
DMSDSK048E INVALID MODE 'mode'  RC=24
DMSDSK054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSDSK062E INVALID * IN FILEID ['fn ft fm']  RC=20
DMSDSK070E INVALID PARAMETER 'parameter'  RC=24
DMSDSK077E END CARD MISSING FROM INPUT DECK  RC=32
DMSDSK078E INVALID CARD IN INPUT DECK  RC=32
DMSDSK104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK RC=100
DMSDSK105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK RC=100
DMSDSK118S ERROR PUNCHING FILE  RC=100
DMSDSK124S ERROR READING CARD FILE  RC=100
DMSDSK205W READER EMPTY OR NOT READY  RC=8
```

DLBL

Use the DLBL command:

- In CMS/DOS, to define DOS and CMS sequential disk files for program input/output; to identify DOS files and libraries; to define and identify VSAM catalogs, clusters, and data spaces; and to identify VSAM, DOS, or CMS files used for VSAM program input/output and access method services functions.

- In CMS, to define and identify VSAM catalogs, clusters, and data spaces; to identify VSAM files used for program input/output; and to identify input/output files for AMSERV.

The format of the DLBL command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│       │                                                                  │
│       │┌                 ┌                  ┐                          ┐│
│ DLBL  ││ddname ⎧mode  ⎫ │CMS fn   ft        │ [(optionA optionB [)]]   ││
│       ││       ⎩DUMMY ⎭ │CMS FILE ddname    │                          ││
│       ││                 └                  ┘                          ││
│       ││                                                               ││
│       ││                 ┌                          ┐                  ││
│       ││ddname ⎧mode  ⎫ │DSN qual1 [qual2...qualn] │                  ││
│       ││       ⎩DUMMY ⎭ │DSN  ?                    │                  ││
│       ││                 └                          ┘                  ││
│       ││                          [(optionA optionB optionC [)]]       ││
│       ││                                                               ││
│       ││ddname CLEAR                                                   ││
│       ││   *                                                           ││
│       │└                                                              ┘│
│       │  optionA:      optionB:      optionC:                            │
│       │  [SYSxxx]      [PERM]        [VSAM ]                             │
│       │                ┌         ┐   ┌      ┐                            │
│       │                │CHANGE   │   │EXTENT│                            │
│       │                │NOCHANGE │   │MULT  │                            │
│       │                └         ┘   └      ┘                            │
│       │                              [CAT catdd]                         │
│       │                              [BUFSP nnnnnn]                      │
└──────────────────────────────────────────────────────────────────────────┘
```

Note: The operands and options of the DLBL command are described below. Usage notes are provided for general usage, followed by additional notes for CMS/DOS users, and then additional notes for OS VSAM users.

where:

ddname specifies a one- to seven-character program ddname (OS) or filename (DOS), or dname (as specified in the FILE parameter of an access method services control statement). An asterisk (*) entered with the CLEAR operand indicates that all DLBL definitions, except those that are entered with the PERM option, are to be cleared.

mode specifies a valid CMS disk mode letter and optionally, filemode number. A letter must be specified; if a number is not specified, it defaults to 1. The disk must be accessed when the DLBL command is issued.

DUMMY specifies that no real I/O is to be performed. A read operation results in an end-of-file condition and a write operation results in a successful return code. DUMMY should not be used for OS VSAM data sets (see Usage Note 3).

CLEAR removes any existing definitions for the specified ddname. Clearing a ddname before defining it ensures that a file definition does not exist and that any options previously defined with that ddname no longer have any effect.

CMS fn ft indicates that this is a CMS file, and the file identifier (fn ft) that follows is a CMS filename and filetype.

FILE ddname is the default CMS file identifier associated with all non-CMS data sets. (See Usage Note 3 for CMS/DOS users.)

DSN indicates that this is a non-CMS file.

? indicates that you are going to enter the data set name interactively. When prompted, you enter the data set name or fileid in its exact form, including embedded blanks, hyphens, or periods.

qual1 [qual2...qualn]
is an OS data set name or DOS file-id. Only data sets named accerding to standard OS conventions may be entered this way; you must omit the periods between qualifiers. (See Usage Note 2.)

Options:

SYSxxx (CMS/DOS only.) indicates the system or programmer logical unit that is associated with the disk on which the disk file resides. The logical unit must have been previously assigned with the ASSGN command. If a DLBL definition is already in effect for the specified ddname, SYSxxx may be omitted; otherwise, it is required.

PERM indicates that this DLBL definition can be cleared only with an explicit CLEAR request. It will not be cleared when the DLBL * CLEAR command line is entered.

All DLBL definitions, including those entered with the PERM option, are cleared as a result of a program abend or HX (halt execution) Immediate command.

CHANGE indicates that any existing DLBL for this ddname is not to be canceled, but that conflicting options are to be overridden and new options merged into the old definition. Both the ddname and the file identifier must be the same in order for the definitions to be merged.

NOCHANGE does not alter any existing DLBL definition for the specified ddname, but creates a definition if none existed.

VSAM indicates that the file is a VSAM data set. This option must be specified for VSAM functions unless the EXTENT, MULT, CAT, or BUFSP options are entered or the ddnames IJSYSCT or IJSYSUC are used.

EXTENT indicates that you are going to use access method services to define a VSAM catalog, data space, or unique cluster and you want to enter extent information.

MULT indicates that you are going to reference an existing multivolume data set and you want to enter the volume specifications.

CAT catdd identifies the VSAM catalog (defined by a previous DLBL definition) which contains the entry for this data set. You must use the CAT option when the VSAM data set you are creating or identifying is not cataloged in the current job catalog. catdd is the ddname in the DLBL definition fcr the catalog.

BUFSP nnnnnn
specifies the number of bytes (in decimal) to be used for I/O buffers by VSAM data management during program execution, overriding the BUFSP value in the ACB for the file. The maximum value for nnnnnn is 999999; embedded commas are not permitted.

Usage Notes

1. To display all of the disk file definitions in effect, enter:

```
dlbl
```

The response will be:

```
ddname DISK fn ft
   .     .   .  .
   .     .   .  .
   .     .   .  .
```

If no DLBL definitions are in effect, the following message is displayed:

```
DMSDLB324I  NO USER DEFINED DLBL'S IN EFFECT
```

2. To enter an OS or DOS file identification on the DLBL command line, it must consist of 1- to 8-character qualifiers separated by periods, with a maximum length of 44 characters, including periods. For example, the file TEST.INPUT.SOURCE.D could be identified as follows:

```
dlbl dd1 c dsn test input source d (options...
```

Or, it may be entered interactively, as follows:

```
dlbl dd1 c dsn ? (options
DMSDLB220R ENTER DATA SET NAME:
test.input.source.d
```

Note that when the data set name is entered interactively, the data set name must be entered in its exact form; when entered on the DLBL command line, the periods must be omitted.

You must use the interactive form to enter a DOS file-id that contains embedded blanks or hyphens.

3. In DOS/VS, a VSAM data set that has been defined as DUMMY is opened with an error code of X'11'. CMS supports the DUMMY operand of the DLBL command in the same manner. OS users should not use the DUMMY operand in CMS, since a dummy data set does not return, on open, an end-of-file indication.

Additional Notes for CMS/DOS Users

1. Each DLBL definition must be associated with a system or programmer logical unit assignment, previously made with an ASSGN command. Specify the SYSxxx option on the first, or only, DLBL definition for a particular ddname. Many DLBL definitions may be associated with the same logical unit. For example:

   ```
   assgn sys100 b
   dlbl dd1 b cms test file1 (sys100
   dlbl dd2 b cms test file2 (sys100
   dlbl dd1 cms test file3
   ```

   is a valid command sequence.

2. The following special ddnames must be used to define DOS private libraries, and must be associated with the indicated logical units:

ddname	Logical Unit	Library
IJSYSSL	SYSSLB	Source statement
IJSYSRL	SYSRLB	Relocatable
IJSYSCL	SYSCLB	Core image

   These libraries must be identified in order to perform librarian functions (with the SSERV, ESERV, DSERV, or RSERV commands) for private libraries; or to link-edit or fetch modules or phases from private relocatable or core image libraries (with the DOSLKED and FETCH commands).

3. Each DOS file has a CMS file identifier associated with it by default; the filename is always FILE and the filetype is always the same as the ddname. For example, if you enter a DLBL command for a DOS file MOD.TEST.STREAM as follows:

   ```
   dlbl test c dsn mod test stream
   ```

   then you can refer to this OS data set as FILE TEST when you use the STATE command:

   ```
   state file test
   ```

   When you enter a DLBL command specifying only a ddname and mode, as follows:

   ```
   dlbl junk a
   ```

   CMS assigns a file identifier of FILE JUNK A1 to the ddname JUNK.

4. The FILEDEF command performs a function similar to that of the DLBL command; you need to use the FILEDEF command in CMS/DOS only:

   - When you want to override a default ddname for an assembler input or output file.

   - When you want to use the MOVEFILE command to process a file.

5. If you use the DUMMY operand, you must have issued an ASSGN command specifying a device type of IGN, or ignore, for the SYSxxx unit specified in the DLBL command, for example,

```
assgn sys003 ign
dlbl test dummy (sys003
```

SPECIFYING VSAM EXTENT INFORMATION: You must specify extent information when you use the access method services control statements DEFINE SPACE, DEFINE MASTERCATALOG, DEFINE USERCATALOG, DEFINE CLUSTER (UNIQUE); or when you use the IMPORT or IMPORTRA functions for a unique file.

When you enter the EXTENT option of the DLBL command, you are prompted to enter the disk extents for the specified file. You must enter extent information in accordance with the following rules:

- For count-key-data devices, you must specify the starting track number and number of tracks for each extent, as follows:

  ```
  19 38
  ```

  This extent allocates 38 tracks, beginning with the 19th track, on a 3330 device.

- For fixed-block devices, you must specify the starting block number and the number of blocks for each extent. The following example allocates 200 blocks, starting at block number 352, on a fixed-block device.

  ```
              352       200
  ```

  Because VSAM rounds the starting block to the next highest cylinder boundary, it is advisable to specify the starting block on a cylinder boundary.

- All count-key-data extents must begin and end on cylinder boundaries, regardless of whether the AMSERV file contains extent information in terms of cylinders, tracks, or records.

- Multiple extent entries may be entered on a single line separated by commas or on different lines. Commas at the end of a line are ignored.

- Multiple extents for the same volume must be entered in numerically ascending order; for example:

  ```
  20 400, 600 80
  ```

  These extents are valid for a 2314 device.

- When you enter multivolume extents, you must specify the mode letter and logical unit associated with each disk that contains extents; extents for each disk must be entered consecutively. For example:

  ```
  assgn sys001 b
  assgn sys002 c
  assgn sys003 d
  dlbl file1 b (extent sys001
  DMSDLB331R ENTER EXTENT SPECIFICATIONS:
  100 60, 400 80, 60 40 d sys003
  200 100 c sys002
  400 100 c sys002
      (null line)
  ```

  specifies extents on disks accessed at modes B, C, and D. These disks are assigned to the logical units SYS001, SYS002, and SYS003. Since B is the mode specified on the DLBL command line, it does not need to be respecified along with the extent information.

- A DASD volume must be mounted, accessed, and assigned for each disk mode referenced in an extent.

When you are finished entering extent information, you must enter a null line to terminate the DLBL command sequence. If you do not, an error may result and you will have to reenter the DLBL command. If you make any error entering the extents, you must reenter all the extent information.

The DLBL command does not check the extents to see whether they are on cylinder boundaries or whether they are entered in the proper sequence. If you do not enter them correctly, the access method services DEFINE functicn will terminate with an error.

CMS assigns sequence numbers to the extents according to the order in which they were entered. These sequence numbers are listed when you use the LISTDS command with the EXTENT option.

In order to display the actual extents that were entered for a VSAM data set at DLBL definition time, the following commands may be entered:

```
    DLBL (EXTENT) or QUERY DLBL EXTENT
```

Either of these commands will provide the following information to the user:

DDNAME The DOS filename or OS ddname.

MODE The CMS disk mode identifying the disk on which the extent resides.

LOGUNIT The DOS logical unit specification (SYSxxx). This operand will be blank for a data set defined while in CMS/CS environment; that is, the SET DOS ON command had not been issued at DLBL definition time.

EXTENT Specifies the relative starting track number and number of tracks for each extent entered for the given dataset ddname.

If no DLBL definitions with extent information are active, the following message is issued:

```
    DMSDLB324I  NO USER DEFINED EXTENTS IN EFFECT
```

IDENTIFYING MULTIVOLUME VSAM EXTENTS: When you want to execute a program or use access method services to reference an existing multivolume VSAM data set, you must use the MULT option on the DLBL command that identifies the file.

When you use the MULT option, you are prompted to enter additional disk mode letters, as follows:

```
    assgn sys001 c
    assgn sys002 d
    assgn sys003 e
    assgn sys004 f
    assgn sys005 g
    dlbl infile c (mult sys001
    DMSDLB330R ENTER VOLUME SPECIFICATIONS:
    d sys002, e sys003 , f sys004
    g sys005
        (null line)
```

The above identifies a file that has extents on disks accessed at modes C, D, E, F, and G. These disks have been assigned to the logical units SYS001, SYS002, SYS003, SYS004, and SYS005. The rules for entering multiple extents are:

- All disks must be mounted, accessed, and assigned when you issue the DLBL command.
- You must nct repeat the mode letter and logical unit of the disk that is entered on the DLBL command line (C in the above example).

- If you enter more than one mode letter and logical unit on a line, they must be separated by commas; trailing commas on a line are ignored.

- A maximum of nine disks may be specified; you do not need to specify them in alphabetical order.

You must enter a null line to terminate the command when you are finished entering extents; if not, an error may result and you must reenter the entire command sequence.

In order to display the volumes on which all multivolume data sets reside, the following commands are issued:

```
DLBL (MULT)  or  QUERY DLBL MULT
```

The following information concerning multiple volume datasets is provided:

DDNAME The DOS filename or OS ddname.

MODE The CMS disk mode identifying one of the disks on which the dataset resides.

LOGUNIT The DOS logical unit specification (SYSxxx). This operand will be blank for a data set defined while in CMS/OS environment; that is, the SET DOS ON command had not been issued at DLBL definition time.

If no DLBL definitions with multiple volume specifications are active, the following message is issued:

```
DMSDLB324I NO USER DEFINED MULTS IN EFFECT
```

USING VSAM CATALOGS: There are two special ddnames you must use to identify a VSAM master catalog and job catalog:

IJSYSCT identifies the master catalog when you initially define it (using AMSERV), and when you begin a terminal session. You should use the PERM option when you define it.

You must assign the logical unit SYSCAT to the disk on which the master catalog resides. If you are redefining a master catalog that has already been identified, you may omit the SYSCAT option on the DLBL command line.

IJSYSUC identifies a job catalog to be used for subsequent AMSERV jobs or VSAM programs.

Any programmer logical unit may be used to assign a job catalog.

Only one VSAM catalog is ever searched when a VSAM function is performed. If a job catalog is defined, you may override it by using the CAT option on the DLBL command for a data set. The following DLBL command sequence illustrates the use of catalogs:

```
assgn syscat c
dlbl ijsysct c dsn mastcat (perm syscat
```

identifies the master catalog, MASTCAT, for the terminal session.

```
     assgn sys010 d
     dlbl ijsysuc d dsn mycat (perm sys010
```

identifies the job (user) catalog, MYCAT, for the terminal session.

```
     assgn sys100 e
     dlbl intest1 e dsn test case (vsam sys100
```

identifies a VSAM file to be used in a program. It is cataloged in the job catalog, MYCAT.

```
     assgn sys101 f
     dlbl cat3 f dsn testcat (cat ijsysct sys101
```

identifies an additional user catalog, which has an entry in the master catalog. Since a job catalog is in use, you must use the CAT option to indicate that another catalog, in this case the master catalog, should be used.

```
     dlbl infile f dsn test input (cat cat3 sys101
```

identifies an input file cataloged in the user catalog TESTCAT, which was identified with a ddname of CAT3 on the DLBL command.

The selection of a VSAM catalog for AMSERV jobs and VSAM programs running in CMS is summarized in Figure 7.

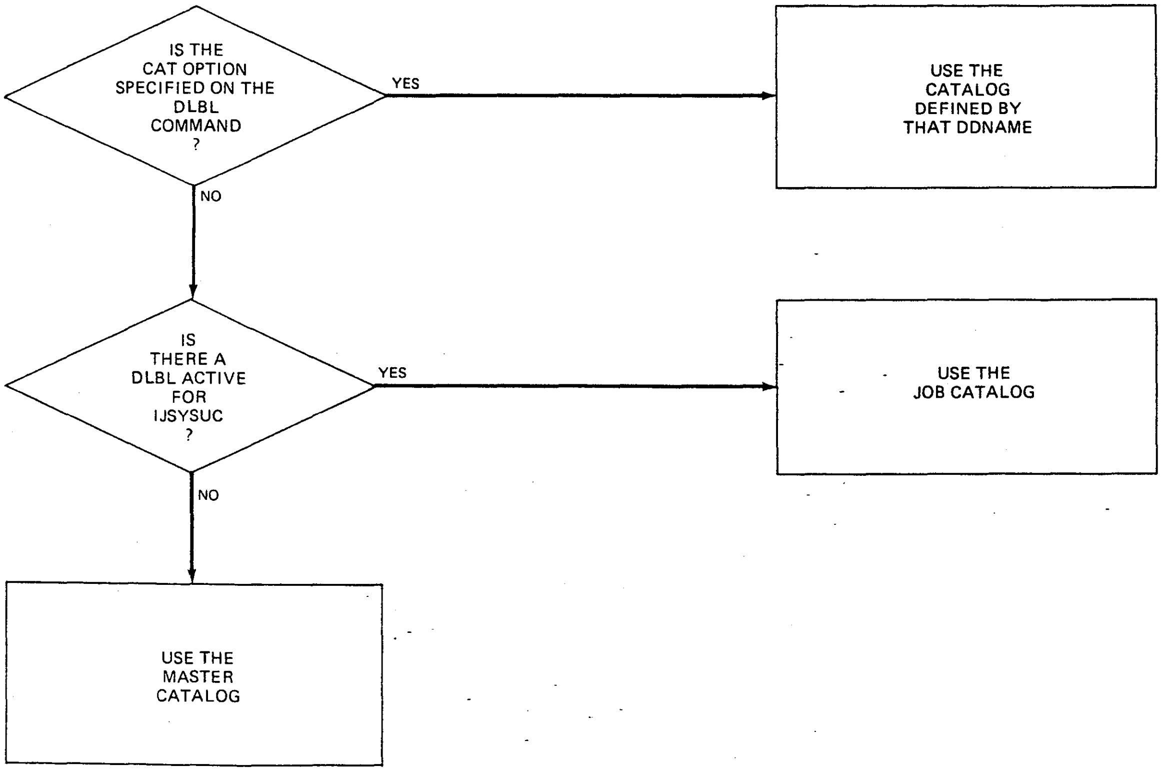


Figure 7. Determining Which VSAM Catalog to Use

Usage Notes for OS VSAM Users

1. You must use the DLBL command to identify all access method services input and output files, and to identify all VSAM input and output files referenced in programs.

   For all other file definitions, including OS or CMS disk files referenced in programs that use VSAM data management, you must use the FILEDEF command.

2. A DLBL ddname may have a maximum of seven characters. If you have ddnames in your programs that are eight characters long, only the first seven characters are processed when the programs are executed in CMS. If you have two ddnames with the same first seven characters and you attempt to execute this program in CMS, you will receive an open error when the second file is opened. You should recompile these programs providing unique seven-character ddnames.

3. If you release a disk for which you have a DLBL definition in effect, you should clear the DLBL definition before you execute a VSAM program or an AMSERV command. CMS checks that all disks for which there are DLBL definitions are accessed, and issues error message DMSSTT069E if any are not.

SPECIFYING VSAM EXTENT INFORMATION: You must specify extent information when you use the access method services control statements DEFINE SPACE, DEFINE MASTERCATALOG, DEFINE USERCATALOG, DEFINE CLUSTER (UNIQUE); or when you use the IMPORT or IMPORTRA functions for a unique file. Space allocation is made only for primary allocation amounts.

When you enter the EXTENT option of the DLBL command, you are prompted to enter the disk extents for the specified file. You must enter extent information in accordance with the following rules:

- For count-key-data devices, you must specify the starting track number and number of tracks for each extent, as follows:

  ```
  19 38
  ```

  This extent allocates 38 tracks, beginning with the 19th track, on a 3330 device.

- For fixed-block devices, you must specify the starting block number and the number of blocks for each extent. The following example allocates 200 blocks, starting at block number 352, on a fixed-block device.

  ```
  352        200
  ```

  Because VSAM rounds the starting block to the next highest cylinder boundary, it is advisable to specify the starting block on a cylinder boundary.
- All count-key-data extents must begin and end on cylinder boundaries, regardless of whether the AMSERV file contains extent information in terms of cylinders, tracks, or records.

- Multiple extent entries may be entered on a single line separated by commas or on different lines. Commas at the end of a line are ignored.

- Multiple extents for the same volume must be entered in numerically ascending order; for example:

  ```
  20 400, 600 80
  ```

  These extents are valid for a 2314 device.

- When you enter multivolume extents, you must specify the mode letter for extents on additional disks; extents for each disk must be entered consecutively. For example:

```
dlbl file1 b (extent
DMSDLB331R ENTER EXTENT SPECIFICATIONS:
100 60, 400 80, 60 40 d
200 100 c
400 100 c
    (null line)
```

specifies extents on disks accessed at modes B, C, and D. Since B is the mode specified on the DLBL command line, it does not need to be respecified along with the extent information.

- A DASD volume must be mounted and accessed for each mode referenced in an extent.

When you are finished entering extent information, you must enter a null line to terminate the DLBL command sequence. If you do not, an error may result and you will have to reenter the entire DLBL command. If you make any error entering the extents, you must reenter all the extent information.

The DLBL command does not check the extents to see if they are on cylinder boundaries or that they are entered in the proper sequence. If you do not enter them correctly, the access method services DEFINE function terminates with an error.

CMS assigns sequence numbers to the extents according to the order in which they were entered. These sequence numbers are listed when you use the LISTDS command with the EXTENT option.

<u>IDENTIFYING</u> <u>MULTIVOLUME</u> <u>VSAM</u> <u>EXTENTS</u>: When you want to execute a program or use access method services to reference an existing multivolume VSAM data set, you must use the MULT option on the DLBL command that identifies the file.

When you use the MULT option, you are prompted to enter additional disk mode letters, as follows:

```
dlbl infile c (mult
DMSDLB330R ENTER VOLUME SPECIFICATIONS:
d, e, f
g
    (null line)
```

The above example identifies a file that has extents on disks accessed at modes C, D, E, F, and G. The rules for entering multiple extents are:

- All disks must be mounted and accessed when you issue the DLBL command.

- You must not repeat the mode letter of the disk that is entered on the DLBL command line (C in the above example).

- If you enter more than one mode letter on a line, they must be separated by commas; trailing commas on a line are ignored.

- A maximum of nine disks may be specified; you do not need to specify them in alphabetical order.

You must enter a null line to terminate the command when you are finished entering extents; if not, an error may result and you must re-enter the entire command sequence.

USING VSAM CATALOGS: There are two special ddnames you must use to identify a VSAM master catalog and job catalog:

IJSYSCT identifies the master catalog, both when you initially define it (using AMSERV) and when you begin a terminal session. You should use the PERM option when you define it.

IJSYSUC identifies a job catalog to be used for subsequent AMSERV jobs or VSAM programs.

Only one VSAM catalog is ever searched when a VSAM function is performed. If a job catalog is defined, you may override it by using the CAT option on the DLBL command for a data set. The following DLBL command sequence illustrates the use of catalogs:

```
    dlbl ijsysct c dsn mastcat (perm
```

identifies the master catalog, MASTCAT, for the terminal session.

```
    dlbl ijsysuc d dsn mycat (perm
```

identifies the job (user) catalog, MYCAT, for the terminal session.

```
    dlbl intest1 e dsn test case (vsam
```

identifies a VSAM file to be used in a program. It is cataloged in the job catalog, MYCAT.

```
    dlbl cat3 dsn testcat (cat ijsysct
```

identifies an additional user catalog, which has an entry in the master catalog. Since a job catalog is in use, you must use the CAT option to indicate that another catalog, in this case the master catalog, should be used.

```
    dlbl infile e dsn test input (cat cat3
```

identifies an input file cataloged in the user catalog TESTCAT, which was identified with a ddname of CAT3 on the DLBL command.

The selection of a VSAM catalog for AMSERV jobs and VSAM programs running in CMS is summarized in Figure 7.

Responses

If the DLBL command is issued with no operands, the current DLBL definitions are displayed at your terminal:

```
    ddname1 device1 [fn1 ft1 fm1 [datasetname1]]
       .       .       .   .   .      .
       .       .       .   .   .      .
       .       .       .   .   .      .
    ddnamen devicen [fnn ftn fmn [datasetnamen]]
```

DMSDLB220R ENTER DATA SET NAME:

This message is displayed when you use the DSN ? form of the DLBL command. Enter the exact DOS or OS data set name.

DMSDLB320I MAXIMUM NUMBER OF DISK ENTRIES RECORDED

This message indicates that nine volumes have been specified for a VSAM data set, which is the maximum allowed under CMS.

DMSDLB321I MAXIMUM NUMBER OF EXTENTS RECORDED

This message indicates that 16 extents on a single disk or minidisk have been specified for a VSAM data space, catalog, or unique data set. This is the maximum number of extents allowed on a minidisk or disk.

DMSDLB322I DDNAME 'ddname' NOT FOUND; NO CLEAR EXECUTED

This message indicates that the clear function was not performed because no DLBL definition is in effect for the ddname.

DMSDLB323I {MASTER|JOB} CATALOG DLBL CLEARED

This message indicates that either the master catalog or job catalog has been cleared as a result of a clear request.

You also receive this message if you issue a DLBL * CLEAR command, and any DLBL definition is in effect for IJSYSCT or IJSYSUC that was not entered with the PERM option.

DMSDLB330R ENTER VOLUME SPECIFICATIONS:

This message prompts you to enter volume specifications for existing multivolume VSAM files. (See "Identifying Multivolume VSAM Extents" in the appropriate usage section.)

DMSDLB331R ENTER EXTENT SPECIFICATIONS:

This message prompts you to enter the data set extent or extents of a new VSAM data space, catalog or unique data set. (See "Specifying VSAM Extent Information" in the appropriate usage section.)

Other Messages and Return Codes

DMSDLB001E NO FILENAME SPECIFIED RC=24
DMSDLB003E INVALID OPTION 'option' RC=24
DMSDLB005E NO '{CAT|BUFSP}' SPECIFIED RC=24
DMSDLB023E NO FILETYPE SPECIFIED RC=24
DMSDLB048E INVALID MODE 'mode' RC=24
DMSDLB050E PARAMETER MISSING AFTER DDNAME RC=24
DMSDLB065E 'option' OPTION SPECIFIED TWICE RC=24
DMSDLB066E 'option' AND 'option' ARE CONFLICTING OPTIONS RC=24
DMSDLB070E INVALID PARAMETER 'parameter' RC=24
DMSDLB086E INVALID DDNAME 'ddname' RC=24
DMSDLB109S VIRTUAL STORAGE CAPACITY EXCEEDED RC=104
DMSDLB221E INVALID DATA SET NAME RC=24
DMSDLB301E 'SYSxxx' NOT ASSIGNED FOR DISK 'fm' RC=36
DMSDLB302E NO SYSXXX OPERAND ENTERED RC=24
DMSDLB304E INVALID OPERAND VALUE 'value' RC=24
DMSDLB305E INCOMPLETE EXTENT RANGE RC=24
DMSDLB306E SYSXXX NOT ASSIGNED FOR 'IGNORE' RC=36
DMSDLB307E CATALOG DDNAME 'ddname' NOT FOUND RC=24
DMSDLB308E 'mode' DISK IN {CMS|NON-CMS} FORMAT; INVALID FOR {NON-CMS|CMS} DATASET RC=24

DOSLIB

Use the DOSLIB command to delete, compact, or list information about the executable phases in a CMS/DOS phase library. The format of the DOSLIB command is:

```
┌──────────────────────────────────────────────────────────────────────┐
| DOSLIB | ( DEL libname phasename1 [...phasenamen]        )           |
|        | )                                               (           |
|        | < COMP libname                                  >           |
|        | )                                               (           |
|        | ( MAP libname [(options...[)]]                  )           |
|        |                                                             |
|        |                          options:                           |
|        |                          ┌      ┐                           |
|        |                          |TERM  |                           |
|        |                          |DISK  |                           |
|        |                          |PRINT |                           |
|        |                          └      ┘                           |
|        |                                                             |
└──────────────────────────────────────────────────────────────────────┘
```

where:

DEL deletes phases from a CMS/DOS phase library. The library is not erased when the last phase is deleted from the library.

COMP compacts a CMS/DOS phase library.

MAP lists certain information about the phases of a DOSLIB. Available information provided is phase name, size, and relative location in the library.

libname is the filename of a CMS/DOS phase library. The filetype must be DOSLIB.

phasename1...phasenamen
is the name of one or more phases that exist in the CMS/DOS phase library.

MAP Options: The following options specify the output device for the MAP function. If more than one option is specified, only the first option is used.

TERM displays the MAP output at the terminal.

DISK writes the MAP output to a CMS disk file with the file identifier of 'libname MAP A5'. If a file with that name already exists, the old file is erased.

PRINT spools the MAP output to the virtual printer.

Usage Notes

1. The CMS/DOS environment does not have to be active when you issue the DOSLIB command.

2. Phases may only be added to a DOSLIB by the CMS/DOS linkage editor as a result of the DOSLKED command.

3. In order to fetch a program phase from a DOSLIB for execution, you must issue the GLOBAL command to identify the DOSLIB. When a FETCH command or dynamic fetch from a program is issued, all current DOSLIBs are searched for the specified phases.

4. If DOSLIBs are very large, or there are many of them to search, program execution is slowed down accordingly. To avoid excessive execution time, you should keep your DOSLIBs small and issue a GLOBAL command specifying only those libraries that you need.

Responses

When you use the TERM option on the DOSLIB MAP command line, the following is displayed:

```
     PHASE      INDEX  BLOCKS
     name1      loc    size
       .          .      .
       .          .      .
       .          .      .
```

Other Messages and Return Codes

```
DMSDSL002E FILE 'fn DOSLIB' NOT FOUND  RC=28
DMSDSL003E INVALID OPTION 'option'  RC=24
DMSDSL013W PHASE 'phase' NOT FOUND IN LIBRARY 'fn DOSLIB fm'  RC=4
DMSDSL014E INVALID FUNCTION 'function'  RC=24
DMSDSL037E DISK 'mode' IS READ/ONLY  RC=36
DMSDSL046E NO LIBRARY NAME SPECIFIED  RC=24
DMSDSL047E NO FUNCTION SPECIFIED  RC=24
DMSDSL069E DISK 'mode' NOT ACCESSED  RC=36
DMSDSL070E INVALID PARAMETER 'parameter'  RC=24
DMSDSL098E NO PHASE NAME SPECIFIED  RC=24
DMSDSL104S ERROR 'nn' READING FILE 'fn DOSLIB fm' FROM DISK  RC=100
DMSDSL105S ERROR 'nn' WRITING FILE 'fn DOSLIB fm' ON DISK  RC=100
DMSDSL213W LIBRARY 'fn DOSLIB fm' NOT CREATED  RC=4
```

DOSLKED

Use the DOSLKED command in CMS/DOS to link-edit TEXT files from CMS disks or object modules from DOS/VSE private or system relocatable libraries and place them in executable form in a CMS phase library (DOSLIB). The format of the DOSLKED command is:

```
┌─────────────────────────────────────────────────────────────────────┐
│           │    ┌       ┐                                            │
│ DOSLKED   │ fn │libname│ [(options...[) ]]                          │
│           │    │fn     │                                            │
│           │    └       ┘                                            │
│           │                 options:                                │
│           │                                ┌     ┐                  │
│           │                                │DISK │                  │
│           │                                │PRINT│                  │
│           │                                │TERM │                  │
│           │                                └     ┘                  │
│           │                                                         │
└─────────────────────────────────────────────────────────────────────┘
```

where:

fn specifies the name of the source file or module to be link-edited. CMS searches for:

1. A CMS file with a filetype of DOSLNK

2. A module in a private relocatable library (if IJSYSRL has been defined)

3. A CMS file with a filetype of TEXT

4. A module in the system relocatable library (if a mode was specified on the SET DOS ON command line)

libname designates the name of the DOSLIB where the link-edited phase is to be written. The filetype is DOSLIB. If libname is not specified, the default is fn. The output filemode of the DOSLIB is determined as follows:

- If libname DOSLIB exists on a read/write disk, that filemode is used and the output is appended to it.

- If fn DOSLNK exists on a read/write disk, libname DOSLIB is written to that disk.

- If fn DOSLNK exists on a read-only extension of a read/write disk, libname DOSLIB is written to the parent disk.

- If none of the above apply, libname DOSLIB is written to your A-disk.

Options: Only one of the following options should be specified. If more than one is specified, only the first entry is used.

DISK writes the DOS/VSE linkage editor map produced by the DOSLKED command on your A-disk into a file with the filename of fn and a filetype of MAP. This is the default option.

PRINT spools the linkage editor map to the virtual printer.

TERM displays the linkage editor map at your terminal.

Note: All error messages are sent to the terminal as well as to the specified device.

Usage Notes

| 1. You can create a CMS file with a filetype of DOSLNK to contain DOS/VSE linkage editor control statements and, optionally, CMS text files.

2. If you want to link-edit a module from a private relocatable library, you must issue an ASSGN command for the logical unit SYSRLB and enter a DLBL command using a ddname of IJSYSRL to identify the library:

```
assgn sysrlb c
dlbl ijsysrl c dsn reloc lib (sysrlb
```

If you have defined a private relocatable library but do not want it to be searched, enter:

```
assgn sysrlb ign
```

to temporarily bypass it.

3. CMS TEXT files may also contain linkage editor control statements INCLUDE, PHASE, and ENTRY. The ACTION statement is ignored when a TEXT file is link-edited.

4. To access modules on the DOS/VS system residence volume, you must have specified the mode letter of the system residence on the SET DOS ON command line:

```
set dos on z
```

5. The search order that CMS uses to locate object modules to be link-edited is:

| a. The specified object module on the DOS/VSE private relocatable library, if one is available

b. CMS disks for a file with the specified filename and with a filetype of TEXT

| c. The specified object module on the DOS/VSE system relocatable library, if it is available

6. When a phase is added to an existing DOSLIB, it is always written at the end of the library. If a phase that is being added has the same name as an existing phase, the DOSLIB directory is updated to point to the new phase. The old phase is not deleted, however; you should issue the DOSLIB command with the COMP option to compress the space.

If you run out of space in a DOSLIB while you are executing the DOSLKED command, you should reissue the DOSLKED command specifying a different DOSLIB, or compress the DOSLIB before attempting to reissue the DOSLKED command.

LINKAGE EDITOR CONTROL STATEMENTS: The CMS/DOS linkage editor recognizes
| and supports the DOS/VSE linkage editor control statements ACTION, PHASE, ENTRY, and INCLUDE. These control statements are described in
| DOS/VSE System Control Statements. The CMS/DOS linkage editor ignores:

- The SVA operand of the PHASE statement
- The F+address form for specifying origin on the PHASE statement
- The BG and Fn operands of the ACTION statement

The S-form of specifying the origin on the PHASE statement corresponds to the CMS user area under CMS/DOS. If a default PHASE statement is required, the origin is assumed to be S. The PBDY operand of the PHASE statement indicates that the phase is link-edited on a 4K page boundary under CMS/DOS as opposed to a 2K page boundary for DOS/VSE.

In DOS/VSE, an ACTION CLEAR control statement clears the unused portion of the core image library to binary zeros. In DOS/VSE the core image library has a defined size, while in CMS/DOS the CMS phase library varies in size, depending on the number of phases cataloged. Therefore, in CMS/DOS an ACTION CLEAR control statement clears the current buffers to binary zeros before loading them; CMS/DOS cannot clear the entire unused portion of the CMS phase library because that portion varies as phases are added to and deleted from the CMS phase library. In CMS/DOS if you want your phases cleared you must issue an ACTION CLEAR control statement each time you add a phase to the CMS phase library.

LINKAGE EDITOR CARD TYPES: The input to the linkage editor can consist of six card types, produced by a language translator or a programmer. These cards appear in the following order:

Card Type	Definition
ESD	External symbol dictionary
SYM	Ignored by linkage editor
TXT	Text
RLD	Relocation list dictionary
REP	Replacement of text made by the programmer
END	End of module

CMS/DOS supports these six card types in the same manner that DOS/VSE does. These card types are described in the *DOS/VSE System Control Statements*.

Responses

When you use the TERM option of the DOSLKED command, the linkage editor map is displayed at the terminal.

2101I INVALID OPERATION IN CONTROL STATEMENT

> This message indicates that a blank card was encountered in the process of link-editing a relocatable module. This message also appears in the MAP file. The invalid card is ignored and processing continues.

Other Messages and Return Codes

```
DMSDLK001E NO FILENAME SPECIFIED  RC=24
DMSDLK003E INVALID OPTION 'option'  RC=24
DMSDLK006E NO READ/WRITE DISK ACCESSED  RC=36
DMSDLK007E FILE 'fn ft fm' IS NOT FIXED, 80-CHAR. RECORDS  RC=32
DMSDLK070E INVALID PARAMETER 'parameter'  RC=24
DMSDLK099E CMS/DOS ENVIRONMENT NOT ACTIVE  RC=40
DMSDLK104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSDLK105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSDLK210E LIBRARY 'library' IS ON READ-ONLY DISK  RC=36
DMSDLK245S ERROR 'nnn' ON PRINTER  RC=100
```

DSERV

Use the DSERV command in CMS/DOS to obtain information that is contained in DOS/VSE private or system libraries. The format of the DSERV command is:

```
┌───────┬──────────────────────────────────────────────────────────────────────┐
│       │ /     ┌               ┌─┐ ┐ \                                          │
│       │ |     │               │nn│ │ |                                         │
│ DSERV │ |  CD │PHASE {name    │12│}│ |  [d2 ...dn] [(options...[)]]            │
│       │ |     └               └─┘ ┘ |                                          │
│       │ <  RD                       >    options:                              │
│       │ |  SD                       |         ┌     ┐                          │
│       │ |  PD                       |         │DISK │                          │
│       │ |  TD                       |         │TERM │                          │
│       │ \  ALL                      /         │PRINT│                          │
│       │                                       └     ┘                          │
│       │                                       [SORT]                           │
└───────┴──────────────────────────────────────────────────────────────────────┘
```

where:

CD
RD
SD
PD
TD
ALL

specifies that information concerning one or more types of directories is to be displayed or printed. The directory types that can be specified are: CD (core image library), RD (relocatable library), SD (source statement library), PD (procedure library), TD (transient directory), and ALL (all directories).

There is no default value. The private libraries take precedence over system libraries.

PHASE name

specifies the name of the phase to be listed. If the phasename ends with an asterisk, all phases that start with the letters preceding the asterisk are listed. This operand is valid only for CD.

nn is the displacement within the phase where the version and level are to be found (the default is 12).

[d2...dn] indicates additional libraries whose directories are to be listed. (See Usage Note 1.)

Options:

DISK writes the output on your CMS A-disk to a file named DSERV MAP A5. This is the default value if TERM or PRINT is not specified.

TERM displays the output at your terminal.

PRINT spools the output to the system printer.

SORT sorts the entries for each library alphamerically; otherwise, the order is the order in which the entries were cataloged.

Usage Notes

1. You may specify more than one directory on DSERV command line; for example:

   ```
   dserv rd sd cd phase $$bopen (term
   ```

   displays the directories of the relocatable and source statement libraries, as well as the entry for the phase $$BOPEN from the core image directory.

   You can specify only one phasename or phasename* at a time, however. If you specify more than one PHASE operand, only the last one entered is listed. For example, if you enter:

   ```
   dserv cd phase cor* phase idc*
   ```

   the file DSERV MAP contains a list of all phases that begin with the characters IDC. The first phasename specification is ignored.

2. If you want to obtain information from the directories of private source statement library directories, relocatable library directories, or core image library directories, the libraries must be assigned and identified (via ASSGN and DLBL commands) when the DSERV command is issued. Otherwise, the system library directories are used. System directories are made available when you specify a mode letter on the SET DOS ON command line.

3. The current assignments for logical units are ignored by the DSERV command; output is directed only to the output device indicated by the option list.

Responses

When you use the TERM option of the DSERV command, the contents of the specified directory are displayed at your terminal.

Other Messages and Return Codes

```
DMSDSV003E INVALID OPTION 'option'  RC=24
DMSDSV021W NO TRANSIENT DIRECTORY  RC=4
DMSDSV022W NO CORE IMAGE DIRECTORY  RC=4
DMSDSV023W NO RELOCATABLE DIRECTORY  RC=4
DMSDSV024W NO PROCEDURE DIRECTORY  RC=4
DMSDSV025W NO SOURCE STATEMENT DIRECTORY  RC=4
DMSDSV026W 'phase' NOT IN LIBRARY  RC=4
DMSDSV027E INVALID DEVICE 'nn'  RC=24
DMSDSV027W NO PRIVATE CORE IMAGE LIBRARY  RC=4
DMSDSV028W NO {PRIVATE|SYSTEM} TRANSIENT DIRECTORY ENTRIES  RC=4
DMSDSV047E NO FUNCTION SPECIFIED  RC=24
DMSDSV065E 'option' OPTION SPECIFIED TWICE  RC=24
DMSDSV066E 'option' AND 'option' ARE CONFLICTING OPTIONS  RC=24
DMSDSV070E INVALID PARAMETER 'parameter'  RC=24
DMSDSV095E INVALID ADDRESS 'address'  RC=24
DMSDSV099E CMS/DOS ENVIRONMENT NOT ACTIVE  RC=40
DMSDSV105S ERROR 'nn' WRITING FILE 'DSERV MAP A5' ON DISK  RC=24
DMSDSV245S ERROR 'nnn' ON PRINTER  RC=100
DMSDSV411S INPUT ERROR CODE 'nn' ON {SYRES|SYSRLB}  RC=24
```

EDIT

Use the EDIT command to invoke the CMS editor to create, modify, and manipulate CMS disk files. Once the editor has been invoked, you may only execute EDIT subcommands and EDIT macro requests, and enter data lines into the disk file. A limited number of CMS commands may be executed in the CMS subset mode, entered from the edit environment.

You can return control to the CMS environment by issuing the EDIT subcommands FILE or QUIT.

For complete details on the EDIT subcommand formats and usage, see "Section 3. EDIT Subcommands and Macros." For tutorial information cn using the CMS editor, including examples, see the *VM/370 CMS User's Guide*. The format of the EDIT command is:

```
| Edit |  fn ft [fm]  [(options...[) ]]
|      |         *
|      |            options:
|      |            [LRECL nn]
|      |            [NODISP]
```

where:

fn ft is the filename and filetype of the file to be created or edited. If a file with the specified filename and filetype does not exist, the CMS editor assumes that you want to create a new file, and after you issue the INPUT subcommand, all data lines you enter become input to the file. If a file with the specified filename and filetype exists, you may issue EDIT subcommands to modify the specified file.

fm is the filemode of the file to be edited, indicating the disk on which the file resides. The editor determines the filemode of the edited file as follows:

Editing existing files: If the file does not reside on your A-disk or its extensions, you must specify fm.

When you specify fm, the specified disk and its extensions are searched. If a file is found on a read-only extension, the filemode of the parent disk is saved; when you issue a FILE cr SAVE subcommand, the modified file is written to the parent disk.

If you specify fm as an asterisk (*) all accessed disks are searched for the specified file.

Creating new files: If you do not specify fm, the new file is written on your A-disk when you issue the FILE or SAVE subcommands.

Options:

LRECL nn is the record length of the file to be created or edited. Use this option to override the default values supplied by the editor, which are determined as follows:

Editing Existing Files: Existing record length is kept regardless of format. If the file has variable-length records and the existing record length is less than the default record length, the default record length is used.

Creating New Files: All new files have a record length of 80, with the following exceptions:

Filetype	LRECL
LISTING	121
SCRIPT,VSBDATA	132
FREEFORT	81

The maximum record length supported by the editor is 160 characters.

NODISP forces a 3270 display terminal into line (typewriter) mode. When the NODISP option is in effect, all subcommands that control the display as a 3270 terminal such as SCROLL, SCROLLUP, and FORMAT (and CHANGE with no operands) are made invalid for the edit session.

Note: It is recommended that the NODISP option always be used when editing on a 3066.

Responses

NEW FILE:

The specified file does not exist.

EDIT:

The edit environment is entered. You may issue any valid EDIT subcommand or macro request.

INPUT:

The input environment is entered by issuing the EDIT subcommands REPLACE or INPUT with no operands. All subsequent input lines are accepted as input to the file.

Other Messages and Return Codes

DMSEDI003E INVALID OPTION 'option' RC=24
DMSEDI024E FILE 'EDIT CMSUT1 fm' ALREADY EXISTS RC=28
DMSEDI029E INVALID PARAMETER 'parameter' IN THE OPTION 'LRECL' FIELD RC=24
DMSEDI044E RECORD LENGTH EXCEEDS ALLOWABLE MAXIMUM RC=32
DMSEDI054E INCOMPLETE FILEID SPECIFIED RC=24
DMSEDI076E ACTUAL RECORD LENGTH EXCEEDS THAT SPECIFIED RC=40
DMSEDI104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK RC=100
DMSEDI105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK RC=100
DMSEDI117S ERROR WRITING TO DISPLAY TERMINAL RC=100
DMSEDI132S FILE 'fn ft fm' TOO LARGE RC=88
DMSEDI143S UNABLE TO LOAD SAVED SYSTEM OR LOAD MODULE RC=40
DMSEDI144S REQUESTED FILE IS IN ACTIVE STATUS

ERASE

Use the ERASE command to delete one or more CMS files from a read/write disk. The format of the ERASE command is:

```
┌──────────────────────────────────────────────────────────────────────┐
│ ERASE   │ fn   ft    fm    [(options...[)]]    options:               │
│         │ *    *     *                                                │
│         │                                                             │
│         │                                         ┌       ┐           │
│         │                                         │Type   │           │
│         │                                         │Notype │           │
│         │                                         └       ┘           │
└──────────────────────────────────────────────────────────────────────┘
```

where:

fn   is the filename of the file(s) to be erased. An asterisk coded in this position indicates that all filenames are to be used. fn must be specified, either with a name or an asterisk.

ft   is the filetype of the file(s) to be erased. An asterisk coded in this position indicates that all filetypes are to be used. This field must be specified, either with a name or an asterisk.

fm   is the filemode of the files to be erased. If this field is omitted, only the A-disk is searched. An asterisk coded in this position indicates that files with the specified filename and/or filetype are to be erased from all read/write disks.

Options:

TYPE   displays at the terminal the file identifier of each file erased.

NOTYPE file identifiers are not displayed at the terminal.

Usage Notes

1. If you specify an asterisk for both filename and filetype you must specify both a filemode letter and number; for example:

```
erase * * a5
```

2. To erase all files on a particular disk, you can use the FORMAT command to reformat it or access the disk using the ACCESS command with the ERASE option.

3. If an asterisk is entered as the filemode, then either the filename or the filetype or both must be specified by name.

Responses

If you specify the TYPE option, the file identification of each file erased is displayed. For example:

```
          erase oldfile temp (type
```

results in the display:

```
          OLDFILE TEMP A1
          R;
```

Other Messages and Return Codes

```
DMSERS002E FILE ['fn [ft [fm]]'] NOT FOUND  RC=28
DMSERS003E INVALID OPTION 'option'  RC=24
DMSERS037E DISK 'mode' IS READ/ONLY  RC=36
DMSERS048E INVALID MODE 'mode'  RC=24
DMSERS054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSERS069E DISK 'mode' NOT ACCESSED  RC=36
DMSERS070E INVALID PARAMETER 'parameter'  RC=24
DMSERS071E ERASE * * [*|mode] NOT ALLOWED  RC=24
DMSERS109T VIRTUAL STORAGE CAPACITY EXCEEDED
```

Note: You can invoke the ERASE command from the terminal, from an EXEC file, or as a function from a program. If ERASE is invoked as a function or from an EXEC file that has the &CONTROL NOMSG option in effect, no error message is issued.

ESERV

Use the ESERV EXEC procedure in CMS/DOS to copy edited DOS/VSE macros from system or private source statement E sublibraries to CMS disk files, or to list de-edited macros. The format of the ESERV command is:

```
┌──────────────────────────────────────────────────────────────┐
| ESERV | fn                                                   |
└──────────────────────────────────────────────────────────────┘
```

where:

fn specifies the filename of the CMS file that contains the ESERV control statements; it must have a filetype of ESERV. The logical unit SYSIPT must be assigned to the disk on which the ESERV file resides. fn is also the filename of the LISTING and MACRO files produced by the ESERV program.

Usage Notes

1. The input file can contain any or all of the ESERV control statements as defined in Guide to the DOS/VSE Assembler.

2. You must have a read/write A-disk accessed when you use the ESERV command.

3. To copy macros from the system source statement library, you must have entered the CMS/DOS environment specifying the mode letter of the DOS/VSE system residence. To copy from a private source statement library, you must assign the logical unit SYSSLB and issue a DLBL command for the ddname IJSYSSL.

4. The output of the ESERV program is directed (as in DOS/VSE) to devices assigned to the logical units SYSLST and/or SYSPCH. If either SYSLST or SYSPCH is not assigned, the following files are created:

Unit	Output File
SYSLST	fn LISTING mode
SYSPCH	fn MACRO mode

   where mode is the mode letter of the disk on which the source file, fn ESERV resides. If fn ESERV is on a read-only disk, the files are written to your A-disk.

   You can override default assignments made by the ESERV EXEC as follows:

   - If you assign SYSIPT to TAPE or READER, the source statements are read from that device.

   - If you assign SYSLST or SYSPCH to another device, the SYSLST or SYSPCH files are written to that device.

5. The ESERV EXEC procedure clears all DLBL definitions, except those entered with the PERM option.

6. If you want to use the ESERV command in an EXEC procedure, you must use the EXEC command (because ESERV is also an EXEC).

7. When you use the ESERV control statements PUNCH or DSPCH, the ESERV program may generate CATAL.S, END, or /* records in the output file. When you add a MACRO file containing these statements to a CMS macro library using the MACLIB command, the statements are ignored and are not read into the MACLIB member.

Responses

None. The CMS ready message indicates that the ESERV program completed execution successfully. You may examine the SYSLST output to verify the results of the ESERV program execution.

Error Messages and Return Codes

```
DMSERV001E NO FILENAME SPECIFIED  RC=24
DMSERV002E FILE 'fn ESERV' NOT FOUND  RC=28
DMSERV006E NO READ / WRITE DISK ACCESSED  RC=36
DMSERV027E INVALID DEVICE ' device ' FOR SYSxxx  RC=28
DMSERV037E DISK 'mode' IS READ ONLY  RC=36
DMSERV070E INVALID ARGUMENT ' argument '  RC=24
DMSERV099E CMS/DOS ENVIRONMENT NOT ACTIVE  RC=40
```

Note: The ESERV EXEC calls other CMS commands to perform certain functions, and so you may, on occasion, receive error messages that occur as a result of those commands.

| Non-CMS error messages produced by the DOS/VSE ESERV program are
| described in the *Guide to the DOS/VSE Assembler*.

EXEC

Use the EXEC command to execute one or more CMS commands or EXEC control statements contained in a specified EXEC file. The format of the EXEC command is:

```
| [EXec] |   fn   [args...]                                              |
```

where:

[EXec] indicates that the EXEC command may be omitted if you are executing the EXEC procedure from the CMS command environment and have not issued the command SET IMPEX OFF.

fn is the filename of a file containing one or more CMS commands and/or EXEC control statements to be executed. The filetype of the file must be EXEC and the file can have either fixed- or variable-length records with a logical record length not exceeding 130 characters. EXEC files can be created with the EDIT command or by a user program. EXEC files created by the CMS editor have, by default, variable-length, 80-character records.

args are any arguments you wish to pass to the EXEC. The arguments are assigned to the special variables &1 through &30 in the order in which they appear in the argument list.

"Section 5. EXEC Control Statements" contains complete descriptions of EXEC control statements, special variables, and built-in functions. For information on designing EXEC procedures and examples of control word usage, see the *VM/370 CMS User's Guide*.

Responses

The amount of information displayed during the execution of an EXEC depends on the setting of the &CONTROL control statement, which by default displays all CMS commands, responses, and error messages. In addition, it displays nonzero return codes from CMS in the format:

```
+++ R(nnnnn) +++
```

where nnnnn is the return code from the CMS command.

For details, see the description of the &CONTROL control statement in "Section 5. EXEC Control Statements."

Messages and Return Codes

If the EXEC interpreter finds an error, it displays the message:

```
DMSEXT072E ERROR IN EXEC FILE filename, LINE nnnn - description
```

The possible errors, and the associated return codes, are:

Description	Return Code
FILE NOT FOUND	801
&SKIP OR &GOTO ERROR	802
BAD FILE FORMAT	803
TOO MANY ARGUMENTS	804
MAX DEPTH OF LOOP NESTING EXCEEDED	805
ERROR READING FILE	806
INVALID SYNTAX	807
INVALID FORM OF CONDITION	808
INVALID ASSIGNMENT	809
MISUSE OF SPECIAL VARIABLE	810
ERROR IN &ERROR ACTION	811
CONVERSION ERROR	812
TOO MANY TOKENS IN STATEMENT	813
MISUSE OF BUILT-IN FUNCTION	814
EOF FOUND IN LOOP	815
INVALID CONTROL WORD	816
EXEC ARITHMETIC UNDERFLOW	817
EXEC ARITHMETIC OVERFLOW	818

Error Message and Return Code

DMSEXC001E NO FILENAME SPECIFIED RC=24

FETCH

Use the FETCH command in CMS/DOS to load an executable phase into storage for execution. The format of the FETCH command is:

```
┌─────────────────────────────────────────────────────────────────────┐
| FETch  | phasename  [(options...[)]]                                |
|        |                                   options:                 |
|        |                                   [START]                  |
|        |                                   [COMP]                   |
|        |                                   [ORIGIN hexloc]          |
└─────────────────────────────────────────────────────────────────────┘
```

where:

phasename is the name of the phase to be loaded into virtual storage. CMS searches for the phase:

- In a DOS/VSE private core image library, if IJSYSCL has been defined
- In CMS DOSLIBs that have been identified with the GLOBAL command
- In the DOS/VSE system core image library, if you specified the mode letter of the DOS/VSE system residence on the SET DOS ON command line

Options:

START specifies that once the phase is loaded into storage, execution should begin immediately.

COMP specifies that when the phase is to be executed, register 1 should contain the address of its entry point. (See Usage Note 5.)

ORIGIN hexloc
fetches the program and loads it at the location specified by hexloc; this location must be in the CMS user area. The location, hexloc, is a hexadecimal number of up to eight characters. (See Usage Note 6.)

Usage Notes

1. If you do not use the START option, FETCH displays a message at your terminal indicating the name of the phase and the storage location of its entry point. At this time, you can set address instruction stops for testing. To continue, issue the START command to initiate execution of the phase just loaded.

2. The fetch routine is also invoked by supervisor call (SVC) instructions 1, 2, 4, or 65. The search order for executable phases is the same as listed above.

3. If you want to fetch a phase from a private core image library, you must issue an ASSGN command for the logical unit SYSCLB and define the library in a DLBL command using the ddname IJSSYCL. For example:

```
assgn sysclb c
dlbl ijsyscl c dsn core image lib (sysclb perm
```

4. Phases fetched from DOS core image libraries must have been link-edited with ACTION REL.

5. CMS uses the COMP option when it fetches the DOS PL/I compiler because that compiler expects register 1 to contain its entry point address. This option is not required when you issue the FETCH command to load your own programs.

   When CMS starts executing a phase that has COMP specified, the DMSLIO740I EXECUTION BEGINS... message is not displayed.

6. The ORIGIN option is used by the CMS/VSAM installation EXEC procedure to load nonsharable modules on a segment boundary. It is not required when you issue the FETCH command to load your own programs, unless you want to load them at a location other than 20000.

7. The FETCH command should only be used with the START command to execute a DOS program. It should not be used with GENMOD to attempt to create an executable CMS module file.

Responses

```
DMSFET710I PHASE 'phase' ENTRY POINT AT LOCATION xxxxxx
```

This message is issued when the START option is not specified. It indicates the virtual storage address at which the phase was loaded.

```
DMSLIO740I EXECUTION BEGINS...
```

This message is issued when the START option is specified; it indicates that program execution has begun.

Other Messages and Return Codes

```
DMSFCH104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSFCH109S VIRTUAL STORAGE CAPACITY EXCEEDED  RC=104
DMSFCH113S DISK (cuu) NOT ATTACHED  RC=100
DMSFCH115E PHASE LOAD POINT LESS THAN 'address'  RC=40
DMSFCH411S INPUT ERROR CODE "nn" ON '{SYSRES|SYSCLB}'  RC=100
DMSFCH777S DOS PARTITION TOO SMALL TO ACCOMMODATE FETCH REQUEST  RC=104
DMSFET003E INVALID OPTION 'option'  RC=24
DMSFET004E PHASE 'phase' NOT FOUND  RC=28
DMSFET029E INVALID PARAMETER  'parameter' IN  THE OPTION  'ORIGIN' FIELD
           RC=24
DMSFET070E INVALID PARAMETER 'parameter'  RC=24
DMSFET098E NO PHASE NAME SPECIFIED  RC=24
DMSFET099E CMS/DOS ENVIRONMENT NOT ACTIVE  RC=40
DMSLIO055E NO ENTRY POINT DEFINED  RC=40
```

FILEDEF

Use the FILEDEF command to establish data definitions for OS ddnames, to define files to be copied with the MOVEFILE command, or to override default file definitions made by the assembler and the OS language processors. The format of the FILEDEF command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
|FIledef| ┌                                                              ┐ |
|       | |(ddname) /Terminal       [(optionA optionD[)]]               \ | |
|       | |< nn    > |                                                   | | |
|       | |(*      ) |PRinter                                            | | |
|       | |          |PUnch         [(optionA[)]]                        | | |
|       | |          |Reader                                             | | |
|       | |          |       ┌              ┌  ┐┐                        | | |
|       | |          |DISK   |fn    ft      |fm|| [(optionA optionB[)]]  | | |
|       | |          |       |FILE ddname   |A1||                        | | |
|       | |          |       └              └  ┘┘                        | | |
|       | |          |                                                   | | |
|       | |          | ┌┌                 ┐┌  ┐┐                         | | |
|       | |          | ||DISK fn   ft     ||fm|| (DSN ?               )  | | |
|       | |          | ||     FILE ddname ||A1|| (DSN qual1 qual2 ...)  | | |
|       | |          | └└                 ┘└  ┘┘                         | | |
|       | |          <                                                   > | |
|       | |          |               [(optionA optionB[)]]               | | |
|       | |          |                                                   | | |
|       | |          |DUMMY          [(optionA[)]]                       | | |
|       | |          |                                                   | | |
|       | |          |       ┌                       ┐                   | | |
|       | |          |TAPn   |LABOFF                 |                   | | |
|       | |          |       |BLP [n]                |                   | | |
|       | |          |       |SL  [n] [VOLID volid]  |                   | | |
|       | |          |       |SUL [n] [VOLID volid]  |                   | | |
|       | |          |       |NL  [n]                |                   | | |
|       | |          |       |NSL filename           |                   | | |
|       | |          |       └                       ┘                   | | |
|       | |          |               [(optionA optionC optionE[)]]       | | |
|       | |          |                                                   | | |
|       | |          \CLEAR                                              / | |
|       | └                                                              ┘ |
|       | optionA:          optionB:           optionC:     optionD:       |
|       | [PERM]            [KEYLEN nnn]       ┌       ┐    ┌       ┐      |
|       | ┌         ┐       ┌            ┐     |7TRACK|     |UPCASE |      |
|       | |CHANGE   |       |XTENT nnnnn |     |9TRACK|     |LOWCASE|      |
|       | |NOCHANGE |       |XTENT 50    |     └       ┘    └       ┘      |
|       | └         ┘       └            ┘     [TRTCH a]                   |
|       | [RECFM a]         [LIMCT nnn]        [DEN den]    optionE:       |
|       | [LRECL nnnnn]     [OPTCD a]                       [LEAVE]        |
|       | ┌             ┐   [DISP MOD]                      [NOEOV]        |
|       | |BLOCK nnnnn  |   [MEMBER membername]                            |
|       | |BLKSIZE nnnnn|   [CONCAT]                                       |
|       | └             ┘   ┌            ┐                                 |
|       |                   |DSORG (PS)  |                                 |
|       |                   |      <PO>  |                                 |
|       |                   |      (DA)  |                                 |
|       |                   └            ┘                                 |
└──────────────────────────────────────────────────────────────────────────┘
```

where:

ddname
nn
* is the name by which the file is referred to in your program. The ddname may be from one to eight alphameric characters, but the first character must be alphabetic or national. If a number nn is specified, it is translated to a FORTRAN data definition name of FTnnF001. An asterisk (*) may be specified with the CLEAR operand to indicate that all file definitions not entered with the PERM option should be cleared.

Devices

TERMINAL is your terminal (terminal I/O must not be blocked).

PRINTER is the spooled printer.

PUNCH is the spooled punch.

READER is the spooled card reader (card reader I/O must not be blocked).

DISK specifies that the virtual I/O device is a disk. As shown in the format, you can choose one of two forms for specifying the DISK operand. Both forms are described in "Using the FILEDEF DISK Operand."

DUMMY indicates that no real I/O takes place for a data set.

TAP[n] is a magnetic tape. The symbolic number of the tape drive, n, can be 1, 2, 3, or 4, representing virtual units 181, 182, 183, and 184, respectively. If n is not specified, TAP2 is the default. You can also specify the type of label processing you want on your tape. Specifying label processing is discussed in "Using the FILEDEF TAPn operand."

CLEAR removes any existing definition for the specified ddname. Clearing a ddname before defining it ensures that a file definition does not exist and that any options previously defined with the ddname no longer have effect.

Options: Whenever an invalid option is specified for a particular device type, an error message is issued. Figure 8 shows valid options for each device type.

Options	OPERANDS READER, PUNCH PRINTER	TERMINAL	TAPn	DISK DUMMY[1]
BLOCK, BLKSIZE	X	X	X	X
CHANGE, NOCHANGE	X	X	X	X
CONCAT				X
DEN			X	
DISP MOD				X
DSORG				X
KEYLEN				X[2]
LEAVE			X	
LIMCT				X[2]
LOWCASE, UPCASE		X		
LRECL	X	X	X	X
MEMBER				X
NOEOV			X	
OPTCD				X[2]
PERM	X	X	X	X
RECFM	X	X	X	X
TRTCH			X[3]	
XTENT				X[2]
7TRACK, 9TRACK			X	

[1] No options may be necessary but all disk options are accepted.

[2] This option is meaningful only for BDAM files.

[3] This option is for 7-track tapes only.

Figure 8. Valid File Characteristics for Each Device Type of the FILEDEF Command

PERM retains the current definition until it either is explicitly cleared or is changed with a new FILEDEF command with the CHANGE option. If PERM is not specified, the definition is cleared when a FILEDEF * CLEAR command is executed.

CHANGE merges the file definitions whenever a file definition already exists for a ddname and a new FILEDEF command specifying the same ddname is issued; the options associated with the two definitions are merged. Options from the original definition remain in effect unless duplicated in the new definition. New options are added to the option list.

NOCHANGE retains the current file definition, if one exists, for the specified ddname.

RECFM a is the record format of the file, where "a" can be one of the following:

a	Meaning
F	fixed length
FB	fixed blocked[1]
V	variable length
VB	variable blocked[1]
U	undefined
FS,FBS	fixed length, standard blocks
VS,VBS	variable length, spanned records
A	ASA print control characters[2]
M	machine print control codes[2]

LRECL nnnnn is the logical record length (nnnnn) of the file, in bytes. LRECL should not exceed 32760 bytes because of OS restrictions.

BLOCK nnnnn
BLKSIZE nnnnn
is the logical block size (nnnnn) of the file, in bytes. BLOCK should not exceed 32760 bytes because of OS restrictions. If both BLOCK and BLKSIZE options are specified, the value of nnnnn for BLOCK is used and BLKSIZE is ignored.

If a CMS file is fixed and has 80-byte CMS records, you should specify RECFM FB BLOCK 800 LRECL 80. Performance can be improved for CMS fixed files if the block size is a multiple of 800.

KEYLEN nnn is the size (nnn) of the key (in bytes). The maximum value accepted is 256.

XTENT nnnnn is the number of records (nnnnn) in the extent for the file. The default is 50. The maximum value is 16,777,215.

LIMCT nnn is the maximum number of extra tracks or blocks (nnn) to be searched. The maximum value is 256.

----

[1] FB and VB should not be used with TERMINAL or READER devices.

[2] A and M may be used with any of the valid RECFM settings (for example, FA, FBA, VA, VBA, etc.) M should not be used with TERMINAL devices.

OPTCD a — is the direct access search processing desired. The variable "a" may be any combination of up to three of the following: (A and R are mutually exclusive.)

Code	DASD Search
A	Actual device addressing
E	Extended search
F	Feedback addressing
R	Relative block addressing

Note: The KEYLEN, XTENT, LIMCT, and OPTCD options should only be used with BDAM files.

DISP MOD — positions the read/write pointer after the last record in the disk file. This option should only be used for output files.

MEMBER membername — allows you to specify the name of a member of an OS partitioned data set; membername is the name of the PDS member.

CONCAT — allows you to assign the same ddname to two or more OS macro libraries so that you can refer to them in a single GLOBAL command.

Any file format options you specify in the first FILEDEF command line remain in effect for subsequently concatenated libraries. For a detailed description of concatenated macro libraries, see "Using OS Macro Libraries" in VM/370 CMS User's Guide.

DSORG {PS | PO | DA} — is the data set organization: physical sequential (PS), partitioned (PO), or direct access (DA).

[7TRACK | 9TRACK] — is the tape setting.

TRTCH a — is the tape recording technique for 7-track tapes. Use the following chart to determine the value of "a" for 7-track tapes.

a	Parity	Converter	Translator
O	odd	off	off
OC	odd	on	off
OT	odd	off	on
E	even	off	off
ET	even	off	on

The default value of TRTCH is OC.

DEN den — is tape density: den can be 200, 556, 800, 1600, or 6250 bpi (bits per inch). If 200 or 556 are specified, 7TRACK is assumed. If 800, 1600, or 6250 are specified 9TRACK is assumed.

UPCASE — translates all terminal input data to uppercase.

LOWCASE — retains all terminal input data as typed in.

LEAVE is only valid for TAPn files that are SUL or SL (standard label). With this option selected, the tape is not moved before label processing. If LEAVE is not specified, tapes with files specified as SL or SUL are rewound and then positioned before the files are processed.

NOEOV is only valid for TAPn files. With NOEOV selected, there is no automatic limited end-of-volume processing when end of tape is sensed on output. See the section "CMS Tape Label Processing" in the _VM/370 CMS User's Guide_ for a description of end-of-volume processing.

Usage Notes

1. If you do not issue a FILEDEF command for an OS input or output file, CMS uses the ddname on the DCB macro to issue the following default file definition:

   ```
   FILEDEF ddname DISK FILE ddname A1
   ```

   See "Usage Notes" under the discussion of the ASSEMBLE command for information on the default file definitions made by the assembler.

2. To identify DOS files for DOS program execution or to identify VSAM data sets for either OS or DOS program execution, you must use the DLBL command.

3. A file definition established with the FILEDEF command remains in effect until explicitly changed or cleared. The system clears file definitions under the following circumstances:

   - When the assembler or any of the language processors are invoked. (Note that FILEDEF definitions entered with the PERM option are not cleared.)

   - When a program abends or when you issue the Immediate command HX to halt command or program execution.

4. The FILEDEF command does not supply default values for LRECL and BLKSIZE. As under OS, if DCB information is unavailable when a file is opened, an open error is issued for the file. The following chart summarizes the results of specifying LRECL and BLKSIZE options.

BLKSIZE	LRECL	Results
Not Specified	Not Specified	If the input file exists on disk, the item length (or item length +4 for variable-length records) becomes the BLKSIZE.
Specified	Not Specified	LRECL=BLKSIZE (or LRECL=BLKSIZE-4, for variable-length records).
Not Specified	Specified	BLKSIZE=LRECL (or BLKSIZE=LRECL+4, for variable-length records).
Specified	Specified	The values specified are used.

   If V or VB is specified for RECFM, LRECL must be at least 4 bytes less than BLKSIZE.

   DOS sequential (SAM) files do not contain BLKSIZE, LRECL, or RECFM specifications. These options must be specified by a FILEDEF command or DCB statement if OS macros are used to access DOS files. Otherwise the defaults, BLKSIZE=32760 and RECFM=U, are assumed. LRECL is not used for RECFM=U files.

5. There is an auxiliary processing option for FILEDEF that is only valid when FILEDEF is executed by an internal program call: this option cannot be entered as a terminal command. The option, AUXPROC addr, allows an auxiliary processing routine to receive control during I/O operations. For details on how to use this option of the FILEDEF command, see *VM/370 System Programmer's Guide*.

6. If a FILEDEF command is issued with a DDNAME that matches a current DDNAME defined by a previous FILEDEF command and the devices are the same, the filename, filetype, filemode, and options previously specified remain in effect, unless respecified by the new FILEDEF command. If the devices are not the same, all previous specifications are removed.

7. If the FILEDEF command is entered with no operands, a list of current definitions is displayed.

Using the FILEDEF DISK Operand

There are two general forms for specifying the DISK operand in a FILEDEF command. If you specify the first form:

```
FILEDEF ddname DISK fn ft [fm]
```

fn and ft (filename and filetype) are assumed to be a CMS fileid. If fm is the filemode of an OS disk, fn and ft are assumed to be the only two qualifiers of an OS data set name. If fm is specified as an asterisk, (*) then the A-disk is assumed.

You cannot use this form unless the OS data set name or DOS file-id conforms to the OS naming convention (1- to 8-byte qualifiers separated by periods, to a maximum of 44 characters, including periods). Also, the data set name can have only two qualifiers; otherwise, you must use the DSN ? or DSN qual1... form. For example, if the OS data set name or DOS file-id is TEST.SAMPLE.MAY, you enter:

```
FILEDEF MINE B1 DSN TEST SAMPLE MAY

-- or --

FILEDEF MINE B1 DSN ?
TEST.SAMPLE.MAY
```

If the OS data set name or DOS file-id is TEST.SAMPLE, then you may enter:

```
FILEDEF MINE DISK TEST SAMPLE B1
```

The second form of the DISK operand is used only with OS data sets and DOS files:

```
               ┌                 ┐ ┌  ┐
FILEDEF ddname │DISK fn   ft     │ │fm│ ⎧DSN ?                 ⎫
               │     FILE ddname │ │A1│ ⎩DSN qual1 [qual2...]  ⎭
               └                 ┘ └  ┘
```

This form allows you to to enter OS and DOS file identifications that do not conform to OS data set naming conventions. The DSN operand corresponds to the DSN parameter on the OS DD (data definition) statement. There are three ways you can specify this form:

- FILEDEF ddname DISK fn ft fm DSN qual1 [qual2...]

  This form of the FILEDEF command associates the CMS filename and filetype you specify with the OS data set name or DOS file-id specified following the DSN operand. Once it is defined, you can refer to the OS data set name or DOS file-id by using the CMS filename and filetype. If you omit DISK, filename, filetype, and filemode, the default values are FILE ddname A1.

- FILEDEF ddname DSN ?

  This form of the FILEDEF command allows you to specify the OS data set name or DOS file-id interactively. Using this form, you can enter an OS data set name or DOS file-id containing embedded special characters such as blanks and hyphens. If you use this form, the default filename and filetype for your file, FILE ddname, is the CMS filename and filetype associated with the OS data set name or DOS file-id. The filemode for this form is always the default, A1.

  To use the interactive DSN operand, you key in DSN ?; CMS then requests that you enter the OS data set name or DOS file-id exactly as it appears in the data set or file. Do nct omit the periods that separate the qualifiers of an OS data set name, but do not insert periods where they do not appear.

  ```
       qual1[.qual2...]
  ```

  where qual1.qual2... are the qualifiers of the OS data set name or DOS file-id. When you use this form, you must code the periods separating the qualifiers.

- FILEDEF ddname mode DSN qual1 [qual2...]

  This form allows you to specify the OS data set name or DOS file-id explicitly. (This form can be used for DOS file-ids only if they comply with the OS naming convention of 1- to 8-byte qualifiers separated by periods, to a maximum of 44 characters, including periods.) Again, the default value for the filename and filetype is FILE ddname. When you use this form, you must omit the periods that separate the qualifiers of the OS data set name. For example, for an OS data set or DOS file named MY.FILE.IN, you enter:

  ```
          FILEDEF ddname B1 DSN MY FILE IN
  ```

All of these forms have many variations, as is apparent from the command format.

Using the FILEDEF TAPn Operand

When you define a tape file with the FILEDEF command, you can specify the type of label processing to be done for the file. You do this by specifying a second operand after the word TAPn. The operands that you may specify and their meanings are:

LABOFF indicates that there is no CMS tape label processing for this tape file. LABOFF is the default. The tape is not positioned if this operand is specified.

BLP indicates that the system is to bypass label processing but that the tape is to be positioned before the file is processed.

SL indicates that you are using IBM standard labels.

SUL indicates that you are using standard user labels (not processed for MOVEFILE).

NL indicates that your tape has no IBM standard labels. (Do not use this operand if your tape has a VOL1 label. A file on it will not be opened.)

NSL indicates that you are using nonstandard labels.

For the operands BLP, SL, and SUL:

n indicates the position of the file on a multifile volume. When n is not specified, the default is 1.

For SL and SUL files:

volid specifies a 1- to 6-character volume serial number to be verified by reading the VOL1 label on the tape. If not specified in FILEDEF, volid may be specified on a LABELDEF command. If specified on both commands, the more recent specification is used. VOLID is only valid for SL or SUL tape files. If VOLID is not specified, the volume label on the tape is not checked.

For the NSL operand:

filename is required for NSL files. It is the filename of a file that contains a routine for processing nonstandard labels. The filename must be that of a TEXT or MODULE file. If you have both a MODULE and TEXT file with this name, the MODULE file is used. MODULE files must be created sc that they start at an address that does not allow them to overlay a user program if they are to be used for NSL routines. See the section "Tape Labels in CMS" in the _VM/370 CMS User's Guide_ for details on writing routines to process nonstandard labels.

You can define a file on tap2 with standard labels by using the the following command:

```
filedef filea tap2 sl volid dept10
```

When this tape file is opened, CMS checks to see that it has a VOL1 label with a volume serial number of dept10.

If you wanted to specify the second file on the same tape, you would use

```
filedef filea tap2 sl 2 volid dept10
```

The same file could be defined as having no labels by using

```
filedef filea tap2 blp 2 filedef filea tap2 nl 2
```

If you use the above specification, your tape must not contain IBM standard labels. NL causes CMS to read your tape at the time you try to open a file on it and checks to see if the tape contains a VOL1 label as its first record. If a VOL1 label is there, CMS does not open your tape file.

If you specify

```
filedef filea tap2 blp 2
```

CMS positions the tape to the second file, but does not check to see whether or not the tape has a label.

Note: If you mount a blank tape and specify NL, the tape will run off the end of the reel. Write a tape mark to prevent this from occurring.

If you wanted to define a tape file with nonstandard labels, use the following command:

```
filedef filea tap2 nsl nonstd
```

The routine NONSTD must exist as a TEXT or MODULE file and be able to process the particular nonstandard labels you are using for your tapes.

If you defined filea with no label parameter at all, for example,

```
        filedef filea tap2
```

there is no label processing or positioning before the data in filea is processed.

It is recommended that you read the section "Tape Labels in CMS" in the _VM/370 CMS User's Guide_ before you write programs that handle labeled tapes.

The LEAVE and NOEOV options are used for tape files only.

LEAVE indicates that a tape containing standard-label files is not to be moved before label processing. Using this option prevents CMS from rewinding the tape and checking the VOL1 label as it otherwise does for SL and SUL files. The command

```
        filedef fileb tap1 sl (leave
```

defines a tape file on tape1 but tells CMS not to position the tape before processing the labels for fileb. Note that you must position the tape properly yourself before using the LEAVE option. LEAVE may be used with SL. SUL, and BLP. However, it has no effect if used with NL. NL tapes are always rewound and positioned before a file on them is opened (even if you specify LEAVE).

The LEAVE option is designed for use with multifile volumes where rewinding and repositioning a tape before each file is processed would be inefficient. You must not move the tape between files if you use this option. Note that for BLP files you can obtain the effect of LEAVE by defining the file as LABOFF rather than BLP.

Using NOEOV, CMS does not do any end-of-tape prccessing on output. If this option is not specified, CMS writes a tape mark after it encounters EOT on output and, for SL and SUL files, also writes an EOV1 label and another tape mark after the first tape mark. The tape is then rewound and unloaded. NOEOV suppresses this limited EOV processing.

Responses

```
ddname1   device1   [filename1  filetype1   filemode1   [datasetname]]
   .         .          .          .           .              .
   .         .          .          .           .              .
   .         .          .          .           .              .
ddnameN   deviceN   [filenameN  filetypeN   filemoden   [datasetname]]
```

A list of current definitions is displayed if the FILEDEF command is entered with no operands.

DMSFLD069I DISK 'mode' NOT ACCESSED

The specified disk is not accessed; the file definition remains in effect. You should access the disk before you attempt to read or write the file.

DMSFLD220R ENTER DATA SET NAME:

A FILEDEF command with the DSN ? operand was entered. Enter the exact OS or DOS file identification, including embedded periods and blanks.

DMSFLD704I INVALID CLEAR REQUEST

A CLEAR request was entered for a file definition that does not exist; no action is taken.

DMSSTT228I USER LABELS BYPASSED ON DATA SET 'data set name'

This message is displayed when you issue a FILEDEF command for an OS data set that contains user labels. The message is displayed the first time you issue the FILEDEF command after accessing the disk on which the data set resides.

Error Messages and Return Codes

```
DMSFLD003E INVALID OPTION 'option'  RC=24
DMSFLD023E NO FILETYPE SPECIFIED  RC=24
DMSFLD027E INVALID DEVICE 'device name'  RC=24
DMSFLD029E INVALID PARAMETER  'parameter' IN  THE OPTION  'option' FIELD
           RC=24
DMSFLD035E INVALID TAPE MODE  RC=24
DMSFLD050E PARAMETER MISSING AFTER DDNAME  RC=24
DMSFLD065E 'option' OPTION SPECIFIED TWICE  RC=24
DMSFLD066E 'option' AND 'option' ARE CONFLICTING OPTIONS  RC=24
DMSFLD070E INVALID PARAMETER 'parameter'  RC=24
DMSFLD221E INVALID DATA SET NAME 'data set name'  RC=24
DMSFLD224E FILEID ALREADY IN USE  RC=24
DMSFLD420E NSL EXIT FILENAME MISSING OR INVALID  RC=24
```

FORMAT

Use the FORMAT command to:

- Initialize a virtual disk (minidisk) for use with CMS files
- Count or reset the number of cylinders on a virtual disk
- Write a label on a virtual disk

The format of the FORMAT command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│            │                                                             │
│   FORMAT   │ cuu mode [nocyl] [ (options...[) ]]                         │
│            │          [noblk]                                            │
│            │                                                             │
│            │              options:                                       │
│            │              ┌              ┌    ┐  ┐                       │
│            │              │Blksize  │ 800│  │                       │
│            │              │              │1024│  │                       │
│            │              │              │2048│  │                       │
│            │              │              │4096│  │                       │
│            │              │              │  1K│  │                       │
│            │              │              │  2K│  │                       │
│            │              │              │  4K│  │                       │
│            │              │              └    ┘  │                       │
│            │              │Noerase                 │                       │
│            │              │Label                   │                       │
│            │              │Recomp                  │                       │
│            │              └                        ┘                       │
└──────────────────────────────────────────────────────────────────────────┘
```

where:

cuu is the virtual device address of the virtual disk to be formatted.

Valid addresses are 001 through 5FF for a virtual machine in basic control mode and 001 through FFF for a virtual machine in extended control mode.

mode is the filemode letter to be assigned to the specified device address. Valid filemode letters are A through Z. This field must be specified. If any other disk is accessed at mode, it is released.

nocyl is the number of cylinders to be made available for use. All available cylinders on the disk are used if the number specified exceeds the actual number available.

noblk is the number of FB-512 blocks to be made available for use. If the number specified exceeds the actual number of blocks on the disk, then all the blocks on the disk are made available for use.

Options:

BLKSIZE specifies the physical DASD block size of the CMS minidisk. The block sizes 1024, 2048, and 4096 may alternately be specified as 1K, 2K, and 4K, respectively. For FB-512 devices, only block sizes 1024, 2048, and 4096 are supported; for CKD (count key data) devices, all block sizes are supported.

NOERASE specifies for FB-512 devices that the permanently formatted FB-512 blocks are not to be cleared to zeros. If not specified, the FB-512 blocks will be cleared. For non-FB-512 devices, this option is ignored.

LABEL writes a label on the disk without formatting the disk. The CMS disk label is written on cylinder 0, track 0, record 3 of the virtual disk or block1 of an FB-512 device. A prompting message requests a six-character disk label (fewer than six characters are left-justified and blanks padded).

RECOMP changes the number of cylinders or FB-512 blocks on the disk that are available to the user. This number becomes the actual number of minidisk cylinders or FB-512 blocks, or the number specified by nocyl/noblk, whichever is less. If nocyl is not specified and the disk is formatted in 800-byte blocks, all cylinders are used. If the disk is formatted in 1K, 2K, or 4K blocks, the maximum number of cylinders initially formatted on the disk is made available to the user.

Usage Notes

1. You can use the FORMAT command with any virtual 3310, 3330, 3340, 3350, 3370, or 2319 device.

2. When you do not specify either the RECOMP or LABEL option, the disk area is initialized by writing a device-dependent number of records (containing binary zeros) on each track. Any previous data on the disk is erased. A read after write check is made as the disk is formatted. For example:

   ```
   format 191 a  25
   ```

   initializes 25 cylinders of the disk located at virtual address 191 in CMS format. The command:

   ```
   format 192 b  25 (recomp)
   ```

   changes the number of cylinders available at virtual address 192 to 25 cylinders, but does not erase any existing data. To change only the label on a disk, you can enter:

   ```
   format 193 c (label)
   ```

   Respond to the prompting message with a six-character label.

3. If you want to format a minidisk for VSAM files, you must use the IBCDASDI program. If you want to format an entire disk, you may use any OS or DOS disk initialization program.

4. Because the FORMAT command requires heavy processor utilization and is heavily I/O bound, system performance may be degraded if there are many users on the system when you use FORMAT.

5. When formatting FB-512 devices, enough blocks of the minidisk area must be formatted to support the CMS disk structure, or message DMS216E will be displayed, and the FORMAT request will be terminated. The number of FB-512 blocks which must be formatted for minidisks of 1K, 2K, and 4K CMS blocksize is 12, 24, and 48, respectively.

Responses

DMSFOR603R FORMAT WILL ERASE ALL FILES ON DISK 'mode(cuu)'. DO YOU WISH TO CONTINUE? (YES|NO):

You have indicated that a disk area is to be initialized: all existing files are erased. This message gives you the option of canceling the execution of the FORMAT command. Reply yes or no.

DMSFOR605R ENTER DISK LABEL:

You have requested that a label be written on the disk. Enter a one- tc six-character label.

DMSFOR705I DISK REMAINS UNCHANGED.

The response to message DMSFOR603R was NO or a null line was entered.

DMSFOR732I {'nnn' CYLINDERS|'nnnnnnnnnn' FB-512 BLOCKS } FORMATTED CN DISK 'mode(cuu)'

The format operation is complete.

DMSFOR733I FORMATTING DISK 'mode'

The disk represented by mode letter 'mode' is being formatted.

```
LABEL CUU M STAT CYL TYPE BLKSIZE FILES BLKS USED-(%) BLKS LEFT BLK TOTAL
label cuu m  R/W nnn type blksize  nnnnn      nnnn- %       nnn    nnnnnn
```

This message provides the status of a disk when you use the RECOMP option. The response is the same as when you issue the QUERY command with the DISK operand.

Other Messages and Return Codes

```
DMSFOR003E INVALID OPTION 'option'  RC=24
DMSFOR005E NO 'option' SPECIFIED  RC=24
DMSFOR017E INVALID DEVICE ADDRESS 'cuu'  RC=24
DMSFOR028E NO DEVICE SPECIFIED  RC=24
DMSFOR037E DISK 'mode[(cuu)]' IS READ/ONLY  RC=36
DMSFOR048E INVALID MODE 'mode'  RC=24
DMSFOR069E DISK 'mode' NOT ACCESSED  RC=36
DMSFOR070E INVALID PARAMETER 'parameter'  RC=24
DMSFOR113S DEVICE 'cuu' NOT ATTACHED  RC=100
DMSFOR114S 'cuu' IS AN UNSUPPORTED DEVICE TYPE
           OR REQUESTED BLKSIZE IS NOT SUPPORTED
           FOR THE DEVICE  RC=88
DMSFOR125S PERMANENT UNIT CHECK ON DISK 'mode(cuu)' RC=100
DMSFOR126S ERROR {READ|WRIT}ING LABEL ON DISK 'mode(cuu)' RC=100
DMSFOR214W CANNOT RECOMPUTE WITHOUT LOSS OF DATA. NO CHANGE  RC=8
DMSFOR216E INSUFFICIENT BLOCKS ON DISK TO SUPPORT
           CMS DISK STRUCTURE  RC=100
```

GENDIRT

Use the GENDIRT command to fill in a CMS auxiliary directory. The auxiliary directory contains the name and location of modules that would otherwise significantly increase the size of the resident directory, thus increasing search time and storage requirements. By using GENDIRT to fill in an auxiliary directory, the file entries for the given command are loaded only when the command is invoked. The format of the GENDIRT command is:

```
| GENDIRT  |  directoryname   [targetmode]                             |
```

__where__:

directoryname
 is the entry point of the auxiliary directory.

targetmode
 is the filemode letter of the disk containing the modules referred to in the directory. The letter is the filemode of the disk containing the modules at execution time, not the filemode of the disk at creation of the directory. At directory creation time, all modules named in the directory being created must be on either the A-disk or a read-only extension; that is, not all disks are searched. The default value for targetmode is S (system disk). It is your responsibility to determine the usefulness of this operand at your installation, and to inform all users whose programs are in auxiliary directories exactly what filemode to specify on the ACCESS command.

__Note__: For information on creating auxiliary directories and for further requirements for using the targetmode option, see the __VM/370 System Programmer's Guide__.

__Messages and Return Codes__

```
DMSGND002W FILE 'fn ft fm' NOT FOUND  RC=4
DMSGND021E ENTRY POINT 'name' NOT FOUND  RC=40
DMSGND022E NO DIRECTORY NAME SPECIFIED  RC=24
DMSGND070E INVALID PARAMETER 'parameter'  RC=24
```

GENMOD

Use the GENMOD command to generate a nonrelocatable (MODULE) file on a CMS disk. The format of the GENMOD command is:

```
 ┌──────────────────────────────────────────────────────────────────────────┐
 │             │                                                            │
|│  Genmod     │ [fn [ MODULE ┌ fm ┐ ]] [(options...[)]]                     │
 │             │              │ A1 │                                         │
 │             │              └    ┘                                         │
 │             │                                                            │
 │             │     options: [ FROM entry1 ]  [ TO entry2 ]                │
 │             │              ┌      ┐  ┌      ┐  ┌   ┐                     │
 │             │              │MAP   │  │STR   │  │OS │                     │
 │             │              │NOMAP │  │NOSTR │  │DOS│                     │
 │             │              └      ┘  └      ┘  │ALL│                     │
 │             │              [SYSTEM]             └   ┘                     │
 └──────────────────────────────────────────────────────────────────────────┘
```

where:

fn is the filename of the MODULE file being created. If fn is not specified, the file created has a filename equal to that of the first entry point in the LOAD MAP.

fm is the filemode of the MODULE file being created. If fm is not specified, A1 is assumed.

Options: If conflicting options are specified, the last one entered is used.

FROM entry1 specifies an entry point or a control section name that represents the starting virtual storage location from which the nonrelocatable copy is generated.

TO entry2 specifies an entry point or a control section name that represents the ending virtual storage location from which the nonrelocatable copy is generated.

MAP includes a load map in the MODULE file. The load map is a variable-length record placed at the end of the load module.

NOMAP specifies that a load map is not to be contained in the MODULE file.

Note: If a module is generated with the NOMAP option, that module cannot later be loaded and started with the CMS LOADMOD and START commands. When NOMAP is specified, the information produced is not sufficient for the START command to execute properly. However, a module generated with the NOMAP option can later be invoked as a command; that is, it can be invoked if its filename is entered.

STR invokes the CMS storage initialization routine when the MODULE file is subsequently loaded (see the LOADMCD command description). This routine frees any storage remaining from a previous program. STR is the default setting if the MODULE is to be loaded at the beginning cf available user storage.

Note: If a program running in the user area calls a transient routine that was generated with the STR option, the user area storage pointers will be reset. This reset condition could cause errors upon return to the original program (for example, when OS GETMAIN/FREEMAIN macros are issued in the user program).

NOSTR indicates that, when the MODULE is loaded, free storage pointers are not reset for any storage currently in use. NOSTR is the default setting if the MODULE file is to be loaded at a location other than the default load address.

SYSTEM indicates that when the MODULE file is subsequently loaded, it is to have a storage protect key of zero.

OS indicates that the program may contain OS macros and, therefore, should be executed only when CMS/DOS is not active.

DOS indicates that the program contains DOS macros; CMS/DCS must be active (that is, SET DOS ON must have been previously invoked) in order for this program to execute. (See Usage Note 2).

ALL indicates that the program:

- Contains CMS macros and must be capable of running regardless of whether CMS/DOS is active or not
- Contains no DOS or OS macros
- Preserves and resets the DOS flag in the CMS nucleus
- Does its own setting of the DOS flags

Note: The ALL option is primarily for use by CMS system programmers. CMS system routines are aware of which environment is active and will preserve and reset the DCS flag in the CMS nucleus.

Usage Notes

1. The GENMOD command is usually invoked following the LOAD command, and possibly the INCLUDE command. For example, the sequence:

   ```
   load myprog
   genmod testprog
   ```

   loads the file MYPROG TEXT into virtual storage and creates a nonrelocatable load module named TESTPROG MODULE. TESTPROG may now be invcked as a user-written command from the CMS environment.

2. The execution of MODULE files created from DOS programs is nct supported and may give unpredictable results. GENMOD is intended for use with the LOAD command, not the FETCH command. Storage initialization for FETCH is different from that for LOAD.

3. Before the file is written, undefined symbols are set to locaticn zero and the common reference control section is initialized. The undefined symbols are not retained as unresolved symbols in the MODULE file. Therefore, once the MODULE file is generated, those references cannot be resolved and may cause unpredictable results during execution.

4. If you load a program into the transient area you should issue the GENMOD command with the STR option. Be careful if the program uses OS GETMAIN or FREEMAIN macros because your program, plus the amount of storage obtained via GETMAIN, cannot exceed two pages (8192 bytes). It is recommended that you do not use GETMAIN macros in programs that execute in the transient area.

5. A transient module (loaded with the ORIGIN TRANS option) that was generated with the SYSTEM option is written on disk as a fixed-length record with a maximum length of 8192 bytes.

6. If you are using FORTRAN under CMS, use FROM MAIN as an option to avoid unpredictable results.

7. If FROM is not specified on the GENMOD command, the starting virtual storage location (entry point) of the module is either the address of fn (if it is an external name) or the entry point determined according to the hierarchy discussed in Usage Note 4 of the LOAD command. This is not necessarily the lowest address loaded. If you have any external references before your START cr CSECT instructions, you must specify the 'FROM entry1' operand on the GENMOD command to load your program properly.

Responses

None.

Messages and Return Codes

```
DMSMOD003E INVALID OPTION 'option'  RC=24
DMSMOD005E NO {FROM|TO} ENTRY SPECIFIED  RC=24
DMSMOD021E ENTRY POINT 'name' NOT FOUND  RC=40
DMSMOD032E INVALID FILETYPE 'ft'  RC=24
DMSMOD037E DISK 'mode' IS READ/ONLY  RC=36
DMSMOD040E NO FILES LOADED  RC=40
DMSMOD070E INVALID PARAMETER 'parameter'  RC=24
DMSMOD084E INVALID USE OF 'FROM' AND 'TO' OPTIONS  RC=24
DMSMOD105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSSTT048E INVALID MODE 'mode'  RC=24
DMSSTT069E DISK 'mode' NOT ACCESSED  RC=36
```

GLOBAL

Use the GLOBAL command to identify which CMS or CMS/DOS libraries are to be searched for macros, copy files, subroutines, or DOS executable phases when processing subsequent CMS commands. The format of the GLOBAL command is:

```
┌──────────────────────────────────────────────────────────────────┐
│  GLobal  │ ( MACLIB ) [libname1 ... libname8]                      │
│          │ < TXTLIB >                                              │
│          │ ( DOSLIB )                                              │
└──────────────────────────────────────────────────────────────────┘
```

<u>where</u>:

MACLIB precedes the specification of macro libraries that are to be searched for macros and copy files during the execution of language processor commands. The macro libraries may be CMS files or OS data sets. If you specify an OS data set, a FILEDEF command must be issued for the data set before you issue the GLOBAL command.

TXTLIB precedes the specification of text libraries to be searched for missing subroutines when the LOAD or INCLUDE command is issued, or when a dynamic load occurs (that is, when an OS SVC 8 is issued).

<u>Note</u>: Subroutines that are called by dynamic load should (1) contain only VCONs that are resolved within the same text library member or (2) be resident in storage throughout the processing of the original CMS LOAD or INCLUDE command. Otherwise, the entry point is unpredictable.

DOSLIB precedes the specification of DOS simulated core image libraries (that is, CMS/DOS phase libraries) to be searched for missing phases. This operand does not apply to system or private core image libraries residing on DOS/VS disks. DOSLIB can be specified regardless of whether the CMS/DOS environment is active or not.

libname1... are the filenames of up to eight libraries. Filetypes must be MACLIB, TXTLIB, and DOSLIB, accordingly. The libraries are searched in the order in which they are named. If no library names are specified, the command cancels the effect of any previous GLOBAL command.

<u>Usage Notes</u>

1. A GLOBAL command remains in effect for an entire CMS session unless it is explicitly canceled or reissued. If a program failure forces you to IPL CMS again, you must reissue the GLOBAL command.

2. There are no default libraries; if you wish to use the same libraries during every terminal session, place the GLOBAL command(s) in your PROFILE EXEC.

3. If you want to use an OS library during the execution of a language processor, you can issue a GLOBAL command to access the library, as long as you have defined the library via the FILEDEF command. If you want to use that library for more than one job, however, you should use the PERM option on the FILEDEF command, since the language processors clear nonpermanent file definitions.

4. You can find out what libraries have been specified by issuing the QUERY command with the MACLIB, TXTLIB, DOSLIB, or LIBRARY operands. (The LIBRARY operand requests a display of all libraries.)

5. For information on creating and/or manipulating CMS libraries, see the discussion of the MACLIB, TXTLIB, and DOSLIB commands.

Responses

None.

Messages and Return Codes

```
DMSGLB002W FILE 'fn ft' NOT FOUND  RC=28
DMSGLB014E INVALID FUNCTION 'function'  RC=24
DMSGLB047E NO FUNCTION SPECIFIED  RC=24
DMSGLB108S MORE THAN 8 LIBRARIES SPECIFIED  RC=88
```

HELP

The HELP command displays descriptions, formats, and parameters of CMS and CP commands and EXECs, and description of CMS and CP messages.

```
┌──────────┬─────────────────────────────────────────────────────────────┐
│          │                                                             │
│          │ ┌                                                        ┐  │
│ HELP     │ │ msg number                                             │  │
│          │ │                                                        │  │
│          │ │                                  ┌                  ┐  │  │
│          │ │ HELP                             │ (option...[) ])  │     │
│          │ │                ┌             ┐   │                  │     │
│          │ │ command name   │ MENU        │   │                  │     │
│          │ │ exec name      │ subcommand  │   │                  │     │
│          │ │ filename       └             ┘   │                  │     │
│          │ │                                  │                  │     │
│          │ │ component name { MENU         }  │                  │     │
│          │ │                { command name }  │                  │     │
│          │ │                { exec name    }  │                  │     │
│          │ │                { subcommand   }  └                  ┘     │
│          │ │                                                     │     │
│          │ └                                                     ┘     │
│          │   options:                                                  │
│          │                                                             │
│          │      ALL                                                    │
│          │      FORM                                                   │
│          │      PARM                                                   │
│          │      DESC                                                   │
│          │                                                             │
└──────────┴─────────────────────────────────────────────────────────────┘
```

where:

msg number
: displays an explanation, reason, system action, user action, and return code as applicable for messages. Message text files for CP and CMS have the form DMKnnnt or DMSnnnt respectively as the file name:

    nnn represents the specific number.
    t represents the message type.

HELP
: displays a description of the function of the HELP command, its syntax, keywords, operands, and options. HELP is the default if no parameters are specified.

component name
: identifies the specific component, such as CP or CMS, that is associated with this request. When you specify component name, you must specify MENU, command, exec name, or subcommand. If it is not specified, it is treated as a CMS command request or a message request.

MENU
: displays a list of those subcommand 'TEXT' files that are available for a component, command, or EXEC that supports subcommands.

command name
: identifies the specific command to be displayed.

exec name
identifies the specific EXEC name to be displayed.

filename
identifies any file that follows the HELP facility file naming conventions and contains HELP 'text' information to be displayed. See "Section 7. HELP Format Words" for information on how to set up these files.

subcommand
identifies the specific subcommand of the command, filename, component, or EXEC specified as the first operand to be displayed.

Options:

ALL
displays all available information.

FORM
displays the syntax form of a command, subcommand, or EXEC.

PARM
displays all applicable keywords, operands, or options for the specified command, subcommand, or EXEC.

DESC
displays a description of the function of the requested command, subcommand, or EXEC.

Usage Notes

1. If you specify more than one option, only the first one is checked for validity.

2. You can enter the CMS immediate command HT when using a line-typing terminal to terminate a successful HELP command request. However, for graphic terminals, you can control the graphic display by typing the following input:

NEXT (or PF10)	to display the next screen; if the current screen is the last, the HELP processing terminates.
BACK (or PF11)	to scroll backwards in the file one screen at a time.
QUIT (or PF12)	to terminate the HELP facility execution.

If program function keys are available, pressing the appropriate key as described above will perform the indicated function. The current screen will be redisplayed and the graphic terminal will sound an audible signal (if your terminal is equipped with that option) under the following conditions:

- A terminal key other than those defined is pressed.
- An incorrect response is entered.
- The current screen being displayed is the first screen of the file and the user requests BACK to view the previous screen.
- The current screen being displayed is the last screen of the file and the user requests NEXT to view a following screen.

If there is only a single screen of information available for display for a given HELP request, the audible signal sounds immediately after execution of the request.

Messages and Return Codes

```
DMSHLP002E INPUT FILE(S) 'fn ft fm' NOT FOUND  RC=28
DMSHLP003E INVALID OPTION 'option'  RC=24
DMSHLP104S ERROR ff READING FILE 'fn ft fm' FROM DISK  RC=104
DMSHLP109E VIRTUAL STORAGE CAPACITY EXCEEDED  RC=104
DMSHLP250S I/O ERROR OR DEVICE ERROR  RC=100
DMSHLP251E HELP PROCESSING ERROR, CODE nnn 'description'  RC=12
           Code  Description

           801   Output line too long.
           802   Format word parameter should be a number.
           803   Invalid format word.
           804   Format word parameter missing.
           805   Invalid format word parameter.
           806   Undent greater than indent.
           807   Excessive or negative space count generated.
           808   Numeric format word parameter is outside valid
                 range.

DMSHLP252E VALID OPTIONS ARE: DESC FORM PARM ALL  RC=28
DMSHLP907T I/O ERROR ON FILE 'fn ft fm'  RC=256
```

INCLUDE

Use the INCLUDE command to read one or more TEXT files (containing relocatable object code) from disk and to load them into virtual storage, establishing the proper linkages between the files. A LOAD command must have been previously issued for the INCLUDE command to produce desirable results. For information on the CMS loader and the handling of unresolved references, see the description of the LOAD command. The format of the INCLUDE command is:

```
┌─────────────────────────────────────────────────────────────────────────┐
| INclude | fn... [(options...[)]]                                        |
|         | options:  ┌        ┐  ┌                ┐  ┌                  ┐ |
|         |           |CLEAR   |  |RESET {entry}   |  |ORIGIN {hexloc}   | |
|         |           |NOCLEAR |  |      {  *  }   |  |       {TRANS }   | |
|         |           └        ┘  └                ┘  └                  ┘ |
|         |                                                               |
|         |   ┌      ┐   ┌       ┐   ┌     ┐   ┌     ┐   ┌      ┐         |
|         |   |MAP   |   |TYPE   |   |INV  |   |REP  |   |AUTO  |         |
|         |   |NOMAP |   |NOTYPE |   |NOINV|   |NOREP|   |NOAUTO|         |
|         |   └      ┘   └       ┘   └     ┘   └     ┘   └      ┘         |
|         |                                                               |
|         |   ┌       ┐                       ┌     ┐                      |
|         |   |LIBE   |   [START]   [SAME]    |DUP  |                      |
|         |   |NOLIBE |                       |NODUP|                      |
|         |   └       ┘                       └     ┘                      |
└─────────────────────────────────────────────────────────────────────────┘
```

where:

fn... are the names of the files to be loaded into storage. Files must have a filetype of TEXT and consist of relocatable object code such as that produced by the OS language processor. If a GLOBAL TXTLIB command has identified one or more TXTLIBs, fn may indicate the name of a TXTLIB member.

Options: If options were specified with a previous LOAD or INCLUDE command, these options (with the exception of CLEAR and ORIGIN) remain set if SAME is specified when INCLUDE is issued. Otherwise, the options assume their default settings. If conflicting options are specified, the last one entered is in effect.

CLEAR clears the load area in storage to binary zeros before the files are loaded.

NOCLEAR does not clear the load area before loading.

RESET {entry}
      { * }

resets the execution starting point previously set by a LOAD or INCLUDE command. If entry is specified, the starting execution address is reset to the specified location. If an asterisk (*) is specified or if the RESET option is omitted, the loader input is searched for control statements. The entry point is selected from the last ENTRY statement encountered or from an assembler- or compiler-produced END statement. If none is found, a default entry point is selected as follows: if an asterisk was specified, the first byte of the first control section loaded by the INCLUDE command becomes the default entry point; if the RESET option was omitted, the entry point defaults to the execution starting point previously set by a LOAD or INCLUDE command.

```
ORIGIN {hexloc}
       {TRANS }
```

begins loading the program at the location specified by hexloc. The variable, hexloc, is a hexadecimal number of up to six characters. If this option is not specified, loading begins at the next available storage location. INCLUDE does not overlay any previously loaded files unless this option is specified and the address given indicates a location within a previously loaded object module. TRANS indicates that the file is loaded into the transient area.

MAP adds information to the load map.

NOMAP does not add any information to the load map.

TYPE displays the load map of the files at the terminal, as well as writing it on the A-disk. This option is valid only if MAP is specified or implied.

NOTYPE does not display the load map at the terminal.

INV writes invalid card images in the LOAC MAP file.

NOINV does not write invalid card images in the LOAC MAP file.

REP writes Replace (REP) statement images in the LOAD MAP file. See the explanation of the CMS LOAD command for a description of the Replace (REP) statement.

NOREP suppresses the writing of Replace (REP) statements in the LOAD MAP file.

AUTO searches your disks for TEXT files to resolve undefined references.

NOAUTO suppresses automatic searching for TEXT files.

LIBE searches the text libraries defined by the GLOBAL command for missing subroutines.

NOLIBE does not search any text libraries for unresolved references.

START begins execution after loading is completed.

SAME retains the same options (except ORIGIN and CLEAR) that were used by a previous INCLUDE or LOAD command. Otherwise, the default setting of unspecified options is assumed. If other options are specified with SAME, they override previously specified options. (See Usage Note 1.)

DUP displays warning messages at your virtual console when a duplicate CSECT is encountered during processing. The duplicate CSECT is not loaded.

NODUP does not display warning messages at your virtual console when duplicate CSECTs are encountered during processing. The duplicate CSECT is not loaded.

Usage Notes

1. If you have specified several nondefault options on the LOAD command, and you want those options to remain in effect, you should use the SAME option when you issue the INCLUDE command; fcr example:

   ```
   include main subi data (reset main map start)
   ```

   brings the files named MAIN TEXT, SUBI TEXT, and DATA TEXT into virtual storage and appends them to files that were previously loaded. Information about these loaded files is added to the LOAD MAP file. Execution begins at entry point MAIN.

   ```
   load myprog (nomap nolibe norep)

   include mysub (map same)
   ```

   During execution of the LOAD command, the file named MYPROG TEXT is brought into real storage. The following options are in effect: NOMAP, NOLIBE, NOREP, NOTYPE, INV, and AUTO. During execution of the INCLUDE command, the file named MYSUE TEXT is appended to MYPROG TEXT. The following options are in effect:

   ```
   MAP, NOLIBE, NOREP, NOTYPE, INV, AUTO
   ```

2. When the INCLUDE command is issued, the loader tables are not reset.

3. For additional information on the CMS loader, see the discussion of the LOAD command, or consult VM/370 CMS User's Guide.

Responses

DMSLIO740I EXECUTION BEGINS...

START was specified with INCLUDE and the loaded program has begun execution. Any further responses are from the program.

INVALID CARD - xxx...xxx

INV was specified with LOAD and an invalid card has been found. The message and the contents of the invalid card (xxx...xxx) are listed in the LOAD MAP file. The invalid card is ignored and loading continues.

Other Messages and Return Codes

```
DMSLGT002I FILE 'fn' TXTLIB NOT FOUND  RC=0
DMSLIO001E NO FILENAME SPECIFIED  RC=24
DMSLIO002E FILE 'fn ft' NOT FOUND  RC=28
DMSLIO003E INVALID OPTION 'option'  RC=24
DMSLIO005E NO 'option' SPECIFIED  RC=24
DMSLIO021E ENTRY POINT 'name' NOT FOUND  RC=40
DMSLIO029E INVALID PARAMETER  'parameter' IN  THE OPTION  'option' FIELD
           RC=24
DMSLIO055E NO ENTRY POINT DEFINED  RC=40
DMSLIO056E FILE 'fn  ft' CONTAINS INVALID  [NAME|ALIAS|ENTRY|ESD] RECORD
           FORMATS  RC=32
DMSLIO099E CMS/DOS ENVIRONMENT ACTIVE  RC=40
DMSLIO104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSLIO105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSLIO109S VIRTUAL STORAGE CAPACITY EXCEEDED  RC=104
DMSLIO116S LOADER TABLE OVERFLOW  RC=104
DMSLIO168S PSEUDO REGISTER TABLE OVERFLOW  RC=104
DMSLIO169S ESDID TABLE OVERFLOW  RC=104
DMSLIO201W THE FOLLOWING NAMES ARE UNDEFINED: RC=4
DMSLIO202W DUPLICATE IDENTIFIER 'identifier'  RC=4
DMSLIO203W "SET LOCATION COUNTER" NAME 'name' UNDEFINED  RC=4
DMSLIO206W PSEUDO REGISTER ALIGNMENT ERROR  RC=4
DMSLIO907T I/O ERROR ON FILE 'fn ft fm'  RC=256
```

LABELDEF

Use the LABELDEF command to specify standard HDR1 and EOF1 tape label description information for CMS, CMS/DOS, and OS simulation. This command is required for CMS/DOS and CMS tape label processing. It is optional for OS simulation but is needed if you want to specify a filename to be checked or the exact data to be written in any field of an output HDR1 and EOF1 label. The format of the LABELDEF command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│ LAbeldef│ {    *    } / CLEAR                                       \     │
│         │ {filename } | ┌            ┐                                 |    │
│         │             | |FID{ ? }|  [VOLID volid] [VOLSER volseq]     |    │
│         │             | |   {fid}|                                    |    │
│         │             | └            ┘                                |    │
│         │             |                                               |    │
│         │             |[FSEQ fseq] [GENN genn] [GENV genv]            |    │
│         │             |                                ┌        ┐     |    │
│         │             <[CRDTE yyddd] [EXDTE yyddd]     |SEC{0}|      >   │
│         │             |[ (options...[) ]]              |   {1}|      |    │
│         │             |Options:                        |   {3}|      |    │
│         │             |                                └        ┘     |    │
│         │             |           ┌          ┐                         |    │
│         │             |[PERM]     |CHANGE    |                         |    │
│         │             |           |NOCHANGE  |                         |    │
│         │             \           └          ┘                        /     │
└──────────────────────────────────────────────────────────────────────────┘
```

where:

* may be specified only with CLEAR. It clears all existing label definitions.

filename is one of the following:

ddname for FILEDEF files (OS simulation).

filename in DTFMT macro (CMS/DOS simulation).

labeldefid specified in the TAPEMAC or TAPPDS command or in the LABID field of the TAPESL macro (can be 1-8 characters).

CLEAR removes a label definition.

LABELDEF filename CLEAR clears only the label definition for that filename.

LABELDEF * CLEAR removes all existing label definitions unless specified as PERM.

FID{ ? }
   {fid} supplies the file (data set for OS) identifier in the tape label. Use the FID ? form if the identifier exceeds 8 characters (up to a maximum of 17) or the identifier contains special characters. The system responds by prompting you to supply the information. If the file identifier does not exceed 8 characters, enter the fileid directly (FID fid).

VOLID volid
is the volume serial number (1-6 numeric characters).

VOLSEQ volseq
is the volume sequence number (1-4 numeric characters).

FSEQ fseq is the file (data set for OS) sequence number in the label (1-4 numeric characters).

GENN genn is the generation number (1-4 numeric characters).

GENV genv is the generation version (1-2 numeric characters).

CRDTE yyddd
is the creation date.

EXDTE yyddd
is the expiration date.

SEC specifies security classification (0 , 1, or 3). See the IBM publication OS/VS Tape Labels, GC26-3795, for the meaning of security classification on tape files. Note that this number has no effect on how the file is processed. It is used only for checking or writing purposes.

options

PERM retains the current definition until it either is explicitly cleared or is changed by a new LABELDEF command with the CHANGE option. If PERM is not specified, the definition is cleared when a LABELDEF * CLEAR command is executed.

CHANGE merges the label definitions whenever a label definition already exists for a filename and a new LABELDEF command specifying the same filename is issued. In this situation, the options associated with the two definitions are merged. Options from the original definition remain in effect unless duplicated in the new definition. New options are added to the option list.

NOCHANGE retains the current label definition, if one exists, for the specified filename.

The following default values are used in output labels when a value is not explicitly specified:

FID For OS simulation, fid is the ddname specified in the FILEDEF command for the file.

For CMS/DOS, fid is the DTFMT symbolic name.

For the CMS TAPESL macro, fid is the LABELDEF specified in the LABID parameter.

VOLID is CMS001.

FSEQ is 0001.

VOLSEQ is 0001.

GENN is blanks.

GENV is blanks.

CRDTE is the date when the label is written.

EXDTE is the date when the label is written.

SEC is 0.

__Usage Notes__:

1. If you want a field checked in an input label, you must specify it on your LABELDEF command for the label. If you do not specify a value for a particular field, this field is not checked at all for input. For output, any field you specify is written in the label exactly as you specify it on the LABELDEF command. If you do not specify a field for output, the default value for that field is written in the label.

   If you write the following LABELDEF command,

   ```
   labeldef filex fid master fseq 2 exdte 78285
   ```

   and use the statement for an input file, only the file identifier, file sequence number, and expiration date in HDR1 labels are checked. Error messages are issued when there fields in the tape label do not match those specified in the LABELDEF statement. If you use the same statement for an output file, the fields leave the following values:

fileid	MASTER
file sequence number	0002
volume sequence number	0001
creation date	date when label is written
expiration date	78285
security	0
volume serial number	CMS001
generation number	blank
generation version	blank

2. If you issue LABELDEF without any operands, a list of all LABELDEFs currently in effect is displayed on your terminal.

3. For OS simulation, a LABELDEF statement may be used as well as a FILEDEF statement for a file. Use of a LABELDEF statement is optional in this case. The statements

   ```
   filedef filez tap1 sl volid vol4
   labeldef filez fid payroll fseq 2 exdte 78300
   ```

   define filez as a labeled tape file on tape 181. The volume serial is VOL4, the fileid is PAYROLL, and the file sequence number is 0002. Expiration date is day 300 in 1978. If you only use the FILEDEF command, you have only defined the VOLID (volume serial number).

4. For CMS and CMS/DOS, a LABELDEF command is required. The command

   ```
   labeldef file14 volid supvol vseq 3
   ```

   defines a tape label with a volume serial of SUPVOL and a volume sequence number of 0003. This LABELDEF statement could be used by a CMS/DOS program containing a DTFMT macro with the form

   ```
   FILE14    DTFMT    ...FILABL=STD...
   ```

   or by a CMS program with a TAPESL macro similar to the following:

   ```
   TAPESL HOUT,181,LABID=FILE14
   ```

   A CMS TAPEMAC command could use the same LABELDEF as follows:

   ```
   tapemac maclib sl file14
   ```

In all three preceding examples, the LABELDEF statement must be issued before the program or command is executed.

5. See the section "Tape Labels in CMS" in the *VM/370 CMS User's Guide* for more details on CMS tape label processing.

Other Messages and Return Codes

```
DMSLBD003E INVALID OPTION-option  RC=24
DMSLBD029E INVALID PARAMETER  'parameter' IN  THE OPTION  'option' FIELD
           RC=24
DMSLBD065E 'option' OPTION SPECIFIED TWICE  RC=24
DMSLBD066E 'option' AND 'option' ARE CONFLICTING OPTIONS RC=24
DMSLBD070E INVALID PARAMETER 'parameter'  RC=24
DMSLBD221E INVALID DATA SET NAME  RC=24
DMSLBD324I NO USER DEFINED LABELDEFS IN EFFECT  RC=20
DMSLBD704I INVALID CLEAR REQUEST  RC=24
```

LISTDS

Use the LISTDS command to list, at your terminal, information about the data sets or files residing on accessed OS or DOS disks, or to display extent or free space information when you want to allocate space for VSAM files. The format of the LISTDS command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
|          | ┌        ┐                                                     |
|          | | ?      | {fm} [(options...[)]]          options:             |
| LISTDS   | |dsname  | {* }                           [FORMAT]             |
|          | └        ┘                                [PDS   ]             |
|          | {fm} (FREE)                               [EXTENT]             |
|          | {* }                                                          |
└──────────────────────────────────────────────────────────────────────────┘
```

where:

? — indicates that you want to enter the OS data set name, DOS file-id, or VSAM data space name interactively. When you enter a question mark (?), CMS prompts you to enter the OS data set name, DOS file-id, or VSAM data space name exactly as it appears on the disk. This fcrm allows you to enter names that contain embedded blanks or hyphens.

dsname — is the OS data set name or DOS file-id or VSAM data space name and takes the form:

qual1 [qual2 qualn]

where qual1, qual2, through qualn are one- to eight-character qualifiers normally separated by periods. Each qualifier must be separated from other qualifiers by blanks when you enter them this way. (See Usage Note 1.)

fm — is the filemode of the disk to be searched for the specified file. If a dsname is not specified, a list of all the files or data sets on the specified disk is displayed.

* — indicates that you want all of your accessed DOS or OS disks searched for the specified data set or file. If a dsname is not specified, a list of all files on all accessed OS and DOS disks is displayed.

Options: The FREE and EXTENT options are mutually exclusive; the FORMAT and PDS options cannot be specified with either FREE or EXTENT.

FREE — requests a display of all free space extents on a specific minidisk or on all accessed DOS and OS disks. If you enter the FREE option, you cannot specify a dsname.

EXTENT
EX — requests a display of allocated extents for a single file or for an entire disk or minidisk. If a dsname is specified, only the extents for that particular file or data set are listed; if fm is specified as *, all disks are searched for extents occupied by that file.

If a dsname is not specified, then a list of all currently allocated extents on the specified disk, or on all disks, is displayed.

FORMAT
FO — requests a display of the date, disk label, filemode, and data set name for an OS data set as well as RECFM, LRECL, BLKSIZE, and DSORG information. For a DOS file, LISTDS displays the date, disk label, filemode, and file-id, but gives no information about the RECFM, LRECL, and BLKSIZE (two blanks appear for each); DSORG is always PS.

PDS — displays the member names of referenced OS partitioned data sets.

For examples of the displays produced as a result of each of these options, see the "Responses" section, below.

Usage Notes

1. If you want to enter an OS or DOS file identification on the LISTDS command line, it must consist of one- to eight-character qualifiers separated by periods. For example, the file TEST.INPUT.SOURCE.D could be listed as follows:

   ```
   listds test input source d *
   ```

   Or, you can enter the name interactively, as follows:

   ```
   listds ? *
   DMSLDS220R ENTER DATA SET NAME:
   test.input.source.d
   ```

   Note that when the data set name is entered interactively, it must be entered in its exact form; when entered on the LISTDS command line, the periods must be omitted.

   You must use the interactive form to enter a DOS file-id that contains embedded blanks or hyphens.

2. You should use the FREE option to determine what free space is available for allocation by VSAM when you are using access method services. For example:

   ```
   listds  * (free
   ```

   requests a display of unallocated extents on all accessed OS or DOS disks. You can then use the EXTENT option on the DLBL command when you define the file for AMSERV.

3. Full disk displays using the FREE option will display free alternate tracks as well as free space extents.

Responses

DMSLDS220R ENTER DATA SET NAME:

This message prompts you to enter the data set name when you use the ? operand on the LISTDS command. Enter the file identification in its exact form. A sample sequence might be:

```
listds ? c
DMSLDS220R ENTER DATA SET NAME:
my.file.test
FM DATA SET NAME
C  MY.FILE.TEST
R;
```

The response shown above following the entry of the data set name is the same as the response given when you enter a data set name on the LISTDS command line.

DMSLDS229I NO MEMBERS FOUND

This message is displayed when you use the PDS option and the data set has no members.

DMSLDS233I NO FREE SPACE AVAILABLE ON 'fm' DISK

This message is displayed when you use the FREE option and there is no free space available on the specified disk.

Responses to the EXTENT Option: A sample response to the EXTENT option is shown below. The headers and the type of information supplied are the same when you request information for a specific file only, or for all disks.

```
listds g (extent

EXTENT INFORMATION FOR 'VTOC' ON 'G' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)     TRACKS
000 VTOC  099 00  1881      099 18   1899           19

EXTENT INFORMATION FOR 'PRIVAT.CORE.IMAGE.LIB' ON 'G' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)    TRACKS
000 DATA  000 01       1     049 18    949        949

EXTENT INFORMATION FOR 'SYSTEM.WORK.FILE.NO.6' ON 'G' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)    TRACKS
000 DATA  050 00     950     051 18    987         38

EXTENT INFORMATION FOR 'COBOL TEST PROGRAM' ON 'G' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)    TRACKS
000 DATA  052 02     990     054 01   1027         38

EXTENT INFORMATION FOR 'DKSQ01A' ON 'G' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)    TRACKS
000 DATA  080 01    1521     081 00   1539         19
```

or for a fixed-block device:

```
EXTENT INFORMATION FOR 'DSQ01A' ON G DISK:
SEQ TYPE  REL-BLK TO REL-BLK   BLOCKS
000 DATA  00500      00550       51
```

where:

SEQ indicates the sequence number assigned this extent when the extents were defined via the DLBL command. CMS assigns the sequence numbers for VSAM data sets; the first extent set has a sequence of 000, the second extent has a sequence of 001, and so on.

TYPE can have the following designations:

Type	Meaning
DATA	Data area extent
VTOC	VTOC extent of the disk
SPLIT	Split cylinder extent
LABEL	User label extent
INDEX	ISAM index area extent
OVFLO	ISAM independent overflow area extent
MODEL	Model data set label in the VTOC. Does not define an extent

CYL-HD(RELTRK) TO CYL-HD(RELTRK)
indicates the cylinder, head, and relative track numbers of the start and end tracks of this extent.

TRACKS indicates the number of tracks in the extent.

| REL-BLK TO REL-BLK
indicates the relative block numbers of the start and end of the extent.

| BLOCKS indicates the number of blocks in the extent.

__Response to the FREE Option__: A sample response to the FREE option is shown below. The same headers and type of information is shown when you request free information for all accessed disks.

```
     listds g (free
     FREESPACE EXTENTS FOR 'G' DISK:
     CYL-HD(RELTRK) TO CYL-HD(RELTRK)    TRACKS
     052 00     988     052 01    989         2
     054 02    1028     080 00   1520       493
     081 01    1540     098 18   1880       341
```

or for a fixed-block device:

```
     listds g (free
     FREESPACE EXTENTS FOR 'G' DISK:
     REL-BLK TO REL-BLK  BLOCKS
       501        1330     830
     10310       29610   19301
     68990       69990    1001
```

__where__:

CYL-HD(RELTRK) TO CYL-HD(RELTRK)
: indicates the cylinder, head and relative track numbers of the starting and ending track in the free extent.

TRACKS indicates the total number of free tracks in the extent.

REL-BLK TO REL-BLK
: indicates the relative block number of the start and end of extents that are free on the fixed-block device.

BLOCKS indicates the total number of blocks contained in each extent.

__Response to the FORMAT and PDS Options__: If you enter the FORMAT and PDS options, you receive information similar to the following:

```
     listds d (fo pds)

RECFM LRECL BLKSI DSORG   DATE     LABEL   FM  DATA SET NAME
  FB    80   800    PO  01/31/75  OSSYS1   D   SYS1.MACLIB
MEMBER NAMES:
ABEND    ATTACH   BLDL     BSP      CLOSE    DCB      DETACH   DEVTYPE
FIND     PUT      READ     WRITE    XDAP
RECFM LRECL BLKSI DSORG   DATE     LABEL   FM  DATA SET NAME
  F     80    80    PS  01/10/75  OSSYS1   D      SAMPLE
```

__Other Messages and Return Codes__

```
DMSLDS002E DATA SET NOT FOUND  RC=28
DMSLDS003E INVALID OPTION 'option'  RC=24
DMSLDS048E INVALID MODE 'mode'  RC=24
DMSLDS069E DISK 'mode' NOT ACCESSED  RC=36
DMSLDS117E INVALID EXTENT FOUND FOR 'data set name' ON 'fm' DISK  RC=24
DMSLDS221E INVALID DATA SET NAME  RC=24
DMSLDS222E I/O  ERROR READING  'data  set  name' FROM  {fm|OS|DOS}  DISK
           RC=28
DMSLDS223E NO FILEMODE SPECIFIED  RC=24
DMSLDS226E NO DATA SET NAME ALLOWED WITH FREE OPTION  RC=24
DMSLDS227W INVALID EXTENT  FOUND FOR  'datasetname' ON  {fm|OS|DOS} DISK
           RC=4
DMSLDS231E I/O ERROR READING VTOC FROM {fm|OS|DOS} DISK  RC=28
```

LISTFILE

Use the LISTFILE command to obtain specified information about CMS files residing on accessed disks. The format of the LISTFILE command is:

```
┌────────────────────────────────────────────────────────────────────────┐
|            |                                                           |
| Listfile   |  ┌   ┌   ┌  ┐┐┐                                           |
|            |  |fn |ft |fm|||  [(options...[)]]                         |
|            |  |*  |*  |* |||                                           |
|            |  └   └   └  ┘┘┘                                           |
|            |                                                           |
|            | options:  ┌         ┐  ┌       ┐  ┌       ┐               |
|            |           |Header   |  |Exec   |  |FName  |               |
|            |           |NOHeader |  |APpend |  |FType  |               |
|            |           └         ┘  └       ┘  |FMode  |               |
|            |                                  |FOrmat |               |
|            |                                  |ALloc  |               |
|            |                                  |Date   |               |
|            |                                  |Label  |               |
|            |                                  └       ┘               |
└────────────────────────────────────────────────────────────────────────┘
```

where:

fn is the filename of the files for which information is to be collected. If an asterisk is coded in this field, all filenames are used. If you code an asterisk preceded by any number of characters, then files that begin with the specified characters are listed.

ft is the filetype of the files for which information is to be collected. If an asterisk is coded in this field, all filetypes are used. If you code an asterisk preceded by any number of characters, then files that begin with the specified characters are listed.

fm is the filemode of the files for which information is to be collected. If this field is omitted, only the A-disk is searched. If an asterisk is coded, all disks are searched.

Output Format Options:

HEADER includes column headings in the listing. HEADER is the default if any of the supplemental information options (FORMAT, ALLOCATE, DATE, or LABEL) are specified. The format of the heading is:

```
FILENAME FILETYPE FM FORMAT LRECL RECS BLOCKS DATE TIME LABEL
```

NOHEADER does not include column headings in the list. NOHEADER is the default if only filename, filetype, or filemode information is requested.

Output Disposition Options:

EXEC creates a CMS EXEC file of 80- or 88-character records (one record for each of the files that satisfies the given file identifier) on your A-disk. An 80-character record file is created unless you specify the LABEL option, in which case an 88-character record file is created. If a CMS EXEC already exists, it is replaced. The header is not included in the file.

APPEND creates a CMS EXEC and appends it to the existing CMS EXEC file. If no CMS EXEC file exists, cne is created.

Information Request Options:

Information Request Options: Only one of these options need be specified. If one is specified, any options with a higher priority are also in effect. If none of the following options are specified, the default information request options are in effect.

Default Information Request Options:

FNAME creates a list containing only filenames. Option priority is 7.

FTYPE creates a list containing only filenames and filetypes. Option priority is 6.

FMODE creates a list containing filenames, filetypes, and filemodes. Option priority is 5.

Supplemental Information Options:

FORMAT includes the record format and logical record length of the of each file in the list. Option priority is 4.

ALLOC includes the amount of disk space that CMS has allocated to the specified file in the list. The quantities given are the number of 800-byte blocks and the number of logical records in the file. Option priority is 3.

DATE includes the date the file was last written in the list.

The form of the date is:

month/day/year hour:minute

for 800-byte block disks, or:

month/day/year hour:minute:second

for all other format sizes.

Option priority is 2.

LABEL includes the label of the disk on which the file resides in the list. Option priority is 1.

Usage Notes

1. If you enter the LISTFILE command with no operands, a list of all files on your A-disk is displayed at the terminal. If you enter:

   ```
   listfile a* f* c
   ```

   you might see the display:

   ```
   AARDVARK  FILE      C5
   ANNA      FILEDATA  C1
   AUTHOR    FLINDEX   C1
   ```

2. If you request any additional information with the supplemental information options, that information is also displayed, along with the header.

3. When you use the EXEC or APPEND option, the CMS EXEC A1 that is created is in the format:

   ```
   pa &2 filename filetype fm ...
   ```

   where column 1 is blank.

   If you use any of the supplemental information options, that information is included in the EXEC file. For information on using CMS EXEC files, see the VM/370 CMS User's Guide.

4. You can invoke the LISTFILE command from the terminal, from an EXEC file, or as a function from a program. If LISTFILE is invoked as a function or from an EXEC file that has the &CONTROL NOMSG option in effect, the DMSLST002E FILE NOT FOUND error message is not issued.

Responses

If the EXEC or APPEND option is not specified, the requested information is displayed at the terminal. Depending on the options specified, or discussed above, the information displayed is:

```
| FILENAME FILETYPE  FM  FORMAT LRECL RECS    BLOCKS   DATE     TIME     LABEL
|                        {F}
     fn    ASSEMBLE  fm  {V}    lrecl norecs  noblks  mm/dd/yy hh:mm:ss volid
     .        .      .   .        .     .       .        .        .       .
     .        .      .   .        .     .       .        .        .       .
     .        .      .   .        .     .       .        .        .       .
```

where:

fn — is the filename of the file.

ft — is the filetype of the file.

fm — is the filemode of the file

{F}
{V} — is the file format: F is fixed-length, V is variable-length.

lrecl — is the logical record length of the largest record in the file.

norecs — is the number of logical records in the file.

noblks — is the number of physical blocks that the file occupies on disk.

| mm/dd/yy is the date (month/day/year) that the file was last updated.

| hh:mm:ss is the time (hours:minutes:seconds) that the file was last updated.

volid is the volume serial number of the virtual disk on which the file resides.

One entry is displayed for each file listed.

Other Messages and Return Codes

```
DMSLST002E FILE NOT FOUND  RC=28
DMSLST003E INVALID OPTION 'option'  RC=24
DMSLST037E DISK 'mode'' IS READ/ONLY  RC=36
DMSLST048E INVALID MODE 'mode'  RC=24
DMSLST066E 'option' and 'option' ARE CONFLICTING OPTIONS  RC=24
DMSLST069E DISK 'mode' NOT ACCESSED  RC=36
DMSLST070E INVALID PARAMETER 'parameter'  RC=24
DMSLST105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
```

LISTIO

Use the LISTIO command in CMS/DOS to display a list of current assignments for system and/or programmer logical units in your virtual machine. The format of the LISTIO command is:

```
┌────────────────────────────────────────────────────────────────────────┐
│         │ ⎛SYS    ⎞                                                    │
│ LISTIO  │ ⎜PROG   ⎟ [ (options...[) ]]                                 │
│         │ ⎨SYSxxx ⎬                   options:                         │
│         │ ⎜A      ⎟                     ┌       ┐                      │
│         │ ⎜UA     ⎟                     │EXEC   │    [STAT]            │
│         │ ⎝ALL    ⎠                     │APPEND │                      │
│         │                               └       ┘                      │
└────────────────────────────────────────────────────────────────────────┘
```

where:

SYS requests a list of the physical devices assigned to all system logical units.

PROG requests a list of the physical devices assigned to programmer logical units SYS000 through SYS241.

SYSxxx requests a display of the physical device assigned to the particular logical unit specified.

A requests a list of only those logical units that have been assigned to physical devices.

UA requests a list of only those logical units that have not been assigned to physical devices; that is, that are unassigned.

ALL requests a list of the physical units assigned to all system and programmer logical units. If no operand is specified, ALL is the default.

Options: The EXEC and APPEND options are mutually exclusive; if both are entered on the command line, the last one entered is in effect.

EXEC erases the existing $LISTIO EXEC file, if one exists, and creates a new one.

APPEND adds new entries to the end of an existing $LISTIO EXEC file. If no $LISTIO EXEC file exists, a new one is created.

STAT lists the status (read-only or read/write) of all disk devices currently assigned.

Usage Notes

1. Logical units are assigned and unassigned with the ASSGN command. For a list of logical units and valid device types, see the discussion of the ASSGN command.

2. The $LISTIO EXEC contains one record for each logical unit listed. The format is:

```
        &1 &2 SYSxxx ⎧ device          ⎫
                     ⎩ mode [status]   ⎭
```

where column 1 is blank.

Responses

Depending on the operands specified, the following is displayed for each unit requested in the LISTIO command:

```
     SYSxxx {device          }
            {mode [status]}
```

where device is the device type (READER, PRINTER, PUNCH, TERMINAL, TAPn, IGN, or UA). If the device is a disk, the one-character mode letter is displayed. If the STAT option is specified, the status (R/O or R/W) is also displayed.

Other Messages and Return Codes

```
DMSLLU003E INVALID OPTION 'option'  RC=24
DMSLLU006E NO READ/WRITE 'A' DISK ACCESSED  RC=36
DMSLLU070E INVALID PARAMETER 'parameter'  RC=24
DMSLLU099E CMS/DOS ENVIRONMENT NOT ACTIVE  RC=40
DMSLLU105S ERROR 'nn' WRITING FILE '$LISTIO EXEC A1' ON DISK  RC=100
```

LOAD

Use the LOAD command to read one or more CMS or OS TEXT files (containing relocatable object code) from disk and to load them into virtual storage, establishing the proper linkages between the files. The format of the LOAD command is:

```
┌───────┬──────────────────────────────────────────────────────────────┐
│ LOAD  │   fn ...   [(options...[)]]                                  │
│       │                                                              │
│       │   options:  ┌        ┐   ┌               ┐   ┌                ┐│
│       │             │CLEAR   │   │RESET {entry}  │   │ORIGIN {hexloc} ││
│       │             │NOCLEAR │   │      {  *  }  │   │       {TRANS } ││
│       │             └        ┘   └               ┘   └                ┘│
│       │                                                              │
│       │       ┌     ┐  ┌      ┐  ┌     ┐  ┌     ┐  ┌      ┐           │
│       │       │MAP  │  │TYPE  │  │INV  │  │REP  │  │AUTO  │           │
│       │       │NOMAP│  │NOTYPE│  │NOINV│  │NOREP│  │NOAUTO│           │
│       │       └     ┘  └      ┘  └     ┘  └     ┘  └      ┘           │
│       │                                                              │
│       │       ┌      ┐            ┌     ┐                             │
│       │       │LIBE  │  [START]   │DUP  │                             │
│       │       │NOLIBE│            │NODUP│                             │
│       │       └      ┘            └     ┘                             │
└───────┴──────────────────────────────────────────────────────────────┘
```

where:

fn... specifies the names of the files to be loaded into storage. The files must have a filetype of TEXT and consist of relocatable object code such as that produced by the OS language processors. If a GLOBAL TXTLIB command has been issued, fn may indicate the name of a TXTLIB member.

Options: If conflicting options are specified, the last one entered is in effect. Options may be overridden or added when you use the INCLUDE command to load additional TEXT files.

CLEAR clears the load area in storage before the object files are loaded. Whole page frames are released; the remainder of storage that is not on a page boundary is set to binary zeros.

NOCLEAR does not clear the load area before loading.

RESET {entry / *}

sets the starting location for the programs currently loaded. The operand, entry, must be an external name (for example, CSECT or ENTRY) in the loaded programs. If RESET is not specified, the default entry point is used. (See Usage Note 4.) If * is entered the results are the same as if the RESET option were omitted.

Note: The RESET option should not be used when loading TEXT files created by any of the following OS/VS language processors under CMS: OS Code and Go FORTRAN, OS FORTRAN IV (G1), OS FORTRAN IV (H) Extended, OS/VS COBOL Compiler and Library, OS Full American National Standard COBOL Version 4 Compiler and Library.

ORIGIN {hexloc / TRANS}

loads the program beginning at the location specified by hexloc; this location must be in the CMS nucleus transient area or in the user area. The location, hexloc, is a hexadecimal number of up to six characters. If TRANS is specified, the file is loaded into the CMS nucleus transient area. If ORIGIN is not specified, loading begins at the first available storage location in the user program area.

Note: Any program loaded into the transient area must have a starting address of X'E000'. See the discussion of the GENMOD command for information on loading programs in the transient area.

MAP writes a load map on your A-disk, named LOAD MAP A5.

NOMAP does not create the LOAD MAP file.

TYPE displays the load map at your terminal, as well as writing it on the A-disk. This option is valid only if the MAP option is in effect.

NOTYPE does not display the load map at the terminal.

INV includes invalid card images in the load map.

NOINV does not include invalid card images in the load map.

REP includes Replace (REP) statements in the load map.

NOREP does not include the Replace (REP) statements in the load map.

AUTO searches your virtual disks for TEXT files to resolve undefined references.

NOAUTO suppresses automatic searching for TEXT files.

LIBE searches the text libraries for missing subroutines. If text libraries are to be searched for TEXT files, they must previously have been defined by a GLOEAL command.

NOLIBE does not search the text libraries for unresolved references.

START executes the program being loaded when loading is completed. LOAD does not normally begin execution of the loaded files. To begin execution immediately upon successful completion of loading, specify START. Execution begins at the default entry point. (See Usage Note 4.)

DUP displays warning messages at your terminal when a duplicate CSECT is encountered during processing. The duplicate CSECT is not loaded. (See Usage Note 3.)

NODUP does not display warning messages at your terminal when duplicate CSECTs are encountered during processing. The duplicate CSECT is not loaded.

Usage Notes

1. You must have a read/write CMS A-disk accessed when you issue the LOAD command; the loader creates a temporary workfile named DMSLDR SYSUT1 and writes it on the A-disk.

2. Unless the NOMAP option is specified, a load map is created on the A-disk each time the LOAD command is issued. A load map is a file that contains the location of control sections and entry points cf files loaded into storage. This load map is named LOAD MAP A5. Each time LOAD is issued, a new LOAD MAP file replaces any previous LOAD MAP file.

   If invalid card images exist in the file or files that are being loaded, they are listed with the message INVALIC CARD in the LOAD MAP file. To suppress this listing in the load map, use the NOINV option.

   If Replace (REP) statements exist in the file being loaded, they are included in the LOAD MAP file. To suppress this listing of REP statements, specify the NOREP option.

   If the ENTRY or LIBRARY control cards are encountered in the file, the load map contains an entry:

   ```
   CONTROL CARD- ...
   ```

   listing the card that was read.

   Mapping of any common areas that exist in the loaded files will occur when the program is prepared for execution by the START or GENMOD command or by the START option of the LOAD or INCLUDE command. An updated load map may be displayed prior to program execution if the START command is issued with the NO option to suppress execution.

3. Duplicate CSECTs (control sections) are bypassed by the loader. Only the first CSECT encountered is physically loaded. The duplicates are not loaded. A warning message is displayed at your terminal if you specified the DUP option. If a section contains an ADCON that references a duplicate CSECT that has not been loaded, that ADCON may be resolved incorrectly.

4. The loader selects the entry point for the loaded program according to the fcllowing hierarchy:

   - Frcm the parameter list on the START command
   - From the last RESET operand in a LOAD or INCLUDE command
   - From the last ENTRY statement in the input
   - From the last LDT statement in the input
   - From the first assembler- or compiler-produced END statement that specifies an entry point if no ENTRY statement is in the input
   - From the first byte of the first control section of the loaded program if there is no ENTRY statement and no assembler- or compiler-produced END statement specifying an entry point

5. The LOAD command should not be used to execute programs containing DOS macros. To link-edit and execute programs in the CMS/DCS environment, use the DOSLKED and FETCH commands.

6. See Figure 9 for an illustration of the loader search order. The loader uses this search order to locate the filename on the LOAD and INCLUDE command lines, as well as in the handling of unresolved references.

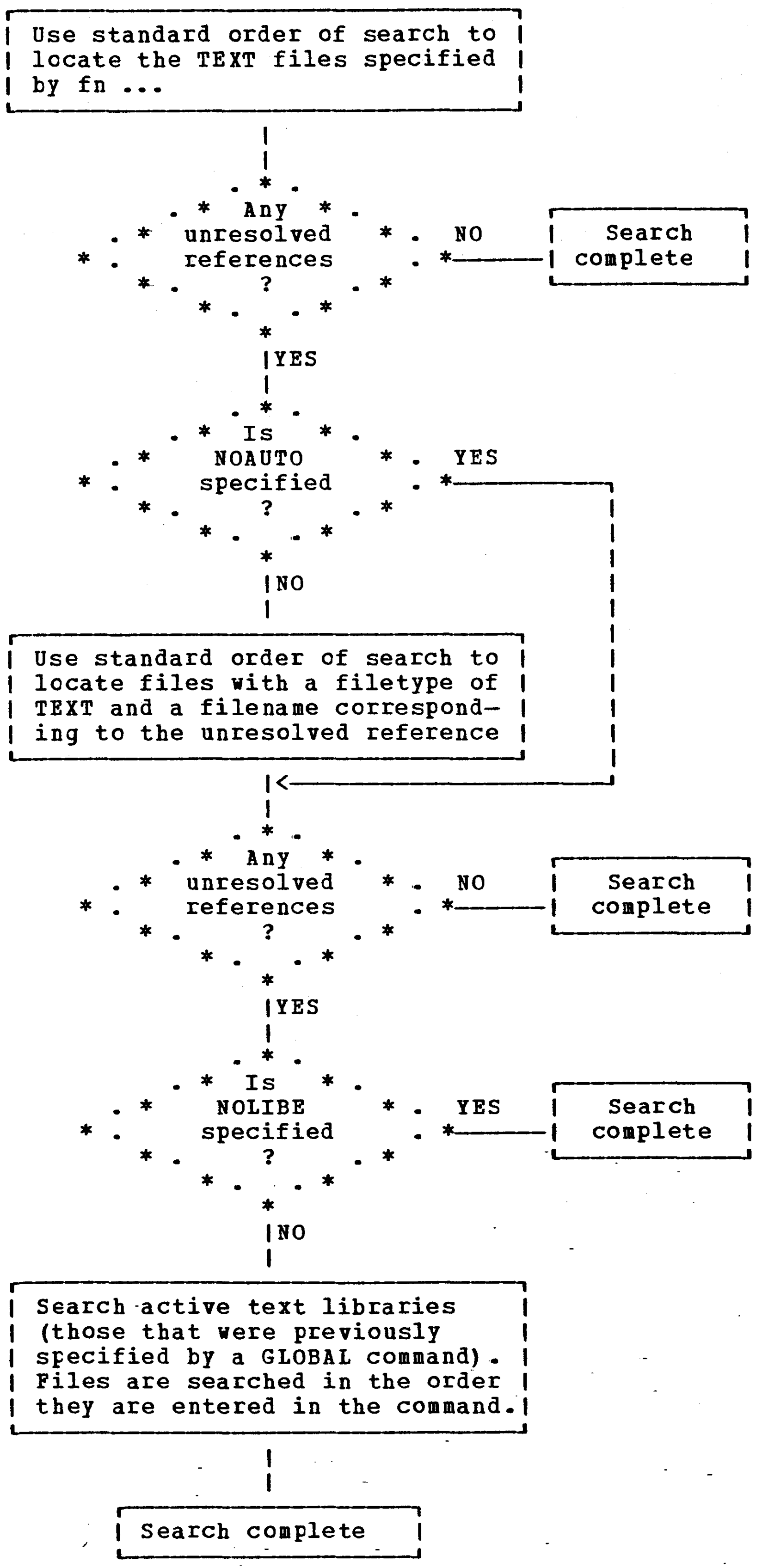

Use standard order of search to locate the TEXT files specified by fn ...
Any unresolved references ?
NO
Search complete
YES
Is NOAUTO specified ?
YES
NO
Use standard order of search to locate files with a filetype of TEXT and a filename corresponding to the unresolved reference
Any unresolved references ?
NO
Search complete
YES
Is NOLIBE specified ?
YES
Search complete
NO
Search active text libraries (those that were previously specified by a GLOBAL command). Files are searched in the order they are entered in the command.
Search complete

Figure 9. Loader Search Order

7. The CMS loader also loads routines called dynamically by OS LINK, LOAD, and XCTL macros. Under certain circumstances, an incorrect entry point may be returned to the calling program. See the *VM/370 CMS User's Guide* for more details.

LOADER CONTROL STATEMENTS

You can add loader control statements to TEXT files either by editing them or by punching real cards and adding them to a punched text deck before reading it into your virtual machine. The seven control cards recognized by the CMS loader are discussed below.

The ENTRY and LIBRARY cards, which are discussed first, are similar to the OS linkage editor control statements ENTRY and LIBRARY. The CMS ENTRY and LIBRARY statements must be entered beginning in column 1.

ENTRY Statement: The ENTRY statement specifies the first instruction to be executed. It can be placed before, between, or after object modules or other control statements. The format of the ENTRY statement is shown in Figure 10. The external name is the name of a control section or an entry name in the input deck. It must be the name of an instruction, not of data.

ENTRY	external name

Figure 10. ENTRY Statement Format

LIBRARY Statement: The LIBRARY statement can be used to specify the never-call function. The never-call function (indicated by an asterisk (*) as the first operand) specifies those external references that are not to be resolved by the automatic library call during any loader step. It is negated when a deck containing the external name referred to is included as part of the input to the loader. The format of the LIBRARY statement is shown in Figure 11. The external reference refers to an external reference that may be unresolved after input processing. It is not to be resolved. Multiple external references within the parentheses must be separated by commas. The LIBRARY statement can be placed before, between, or after object decks or other control statements.

LIBRARY	* (external reference)

Figure 11. LIBRARY Statement Format

Loader Terminate (LDT) Statement: The LDT statement is used in a text library as the last record of a member. It indicates to the loader that all records for that member were processed. The LDT statement can contain a name to be used as the entry point for the loaded member. The LDT statement has the format shown in Figure 12.

Column	Contents
1	X'02' (12-2-9 punch). Identifies this as a loader control statement.
2-4	LDT — identifies type of statement.
5-16	Not used.
17-24	Blank or entry name (left-justified and padded with blanks to eight characters).
25	Blank.
26-33	May contain information specified on a SETSSI card processed by the TXTLIB command.
34-80	Not used.

Figure 12. LDT Statement Format

Include Control Section (ICS) Statement: The ICS statement changes the length of a specified control section or defines a new control section. It should be used only when REP statements cause a control section to be increased in length. The format of an ICS statement is shown in Figure 13. An ICS statement must be placed at the front of the file or TEXT file.

Column	Contents
1	X'02' (12-2-9 punch). Identifies this as a loader control statement.
2-4	ICS — identifies the type of load statement.
5-16	Blank.
17-22	Control section name — left-justified in these columns.
23	Blank.
24	, (comma).
25-28	Hexadecimal length in bytes of the control section. This must not be less than the actual length of the previously specified control section. It must be right-justified in columns with unused leading columns filled with zeros.
29	Blank.
30-72	May be used for comments or left blank.
73-80	Not used by the loader. You may leave these columns blank or insert program identification for your own convenience.

Figure 13. ICS Statement Format

Set Location Counter (SLC) Statement: The SLC statement sets the location counter used with the loader. The file loaded after the SLC statement is placed in virtual storage beginning at the address set by this SLC statement. The SLC statement has the format shown in Figure 14. It sets the location counter in one of three ways:

1. With the absolute virtual address specified as a hexadecimal number in columns 7-12.

2. With the symbolic address already defined as a program name or entry point. This is specified by a symbolic name punched in columns 17-22.

3. If both a hexadecimal address and a symbolic name are specified, the absolute virtual address is converted to binary and added to the address assigned to the symbolic name; the resulting sum is the address to which the loader's location counter is set. For example, if 0000F8 was specified in columns 7-12 of the SLC card image and GAMMA was specified in columns 17-22, where GAMMA has an assigned address of 006100 (hexadecimal), the absolute address in columns 7-12 is added to the address assigned to GAMMA giving a total of 0061F8. Thus, the location counter would be set to 0061F8.

Column	Contents
1	X'02' (12-2-9 punch). Identifies this as a loader control statement.
2-4	SLC — identifies the type of load statement.
5-6	Blank.
7-12	Hexadecimal address to be added to the value of the symbol, if any, in columns 17-22. It must be right-justified in these columns, with unused leading columns filled with zeros.
13-16	Blank.
17-22	Symbolic name whose assigned location is used by the loader. Must be left-justified in these columns. If blank, the address in the absolute field is used.
23	Blank.
24-72	May be used for comments or left blank.
73-80	Not used by the loader. You may leave these columns blank or insert program identification for your own convenience.

Figure 14. SLC Statement Format

Replace (REP) Statement: A REP statement allows instructions and constants to be changed and additions made. The REP statement must be punched in hexadecimal code. The format of a REP statement is shown in Figure 15. The data in columns 17-70 (excluding the commas) replaces what has already been loaded into virtual storage, beginning at the address specified in columns 7-12. REP statements are placed in the file either (1) immediately preceding the last statement (END statement) if the text deck does not contain relocatable data such as address constants, or (2) immediately preceding the first RLD (relocatable dictionary) statement if there is relocatable data in the text deck. If additions made by REP statements increase the length of a control section, an ICS statement, which defines the total length of the control section, must be placed at the front of the deck.

Column	Contents
1	X'02' (12-2-9 punch). Identifies this as a loader control statement.
2-4	REP — identifies the type of load statement.
5-6	Blank.
7-12	Hexadecimal starting address of the area to be replaced as assigned by the assembler. It must be right-justified in these columns with unused leading columns filled with zeros.
13-14	Blank.
15-16	ESID (External Symbol Identification) — the hexadecimal number assigned to the control section in which replacement is to be made. The LISTING file produced by the compiler or assembler indicates this number.
17-70	A maximum of 11 four-digit hexadecimal fields, separated by commas, each replacing one previously loaded halfword (two bytes). The last field must not be followed by a comma.
71-72	Blank.
73-80	Not used by the loader. This field may be left blank or program identification may be inserted.

Figure 15. REP Statement Format

Set Page Boundary (SPB) Statement: An SPB statement instructs the loader to update the location counter to point to the next page boundary. The SPB statement has the format shown in Figure 16.

Column	Contents
1	X'02' (12-2-9 punch). Identifies this as a loader control statement.
2-4	SPB — identifies the type of load statement.
5-80	May be used for comments or left blank.

Figure 16. SPB Statement Format

Responses

DMSLIO740I EXECUTION BEGINS...

START was specified with LOAD and the loaded program starts execution. Any further responses are from the program.

INVALID CARD - xxx...xxx

INV was specified with LOAD and an invalid statement was found. The message and the contents of the invalid statement (xxx...xxx) are listed in the file LOAD MAP. The invalid statement is ignored and loading continues.

Other Messages and Return Codes

```
DMSLGT002I FILE 'fn TXTLIB' NOT FOUND  RC=0
DMSLIO001E NO FILENAME SPECIFIED  RC=24
DMSLIO002E FILE 'fn ft' NOT FOUND  RC=28
DMSLIO003E INVALID OPTION 'option'  RC=24
DMSLIO005E NO 'option' SPECIFIED  RC=24
DMSLIO021E ENTRY POINT 'name' NOT FOUND  RC=40
DMSLIO029E INVALID PARAMETER 'parameter' IN THE OPTION 'option' FIELD  RC=24
DMSLIO055E NO ENTRY POINT DEFINED  RC=40
DMSLIO056E FILE 'fn  ft' CONTAINS INVALID  [NAME|ALIAS|ENTRY|ESD] RECORD
           FORMATS  RC=32
DMSLIO099E CMS/DOS ENVIRONMENT ACTIVE  RC=40
DMSLIO104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSLIO105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSLIO109S VIRTUAL STORAGE CAPACITY EXCEEDED  RC=104
DMSLIO116S LOADER TABLE OVERFLOW  RC=104
DMSLIO168S PSEUDO REGISTER TABLE OVERFLOW  RC=104
DMSLIO169S ESDID TABLE OVERFLOW  RC=104
DMSLIO201W THE FOLLOWING NAMES ARE UNDEFINED:  RC=4
DMSLIO202W DUPLICATE IDENTIFIER 'identifier'  RC=4
DMSLIO203W "SET LOCATION COUNTER" NAME 'name' UNDEFINED  RC=4
DMSLIO206W PSEUDO REGISTER ALIGNMENT ERROR  RC=4
DMSLIO907T I/O ERROR ON FILE 'fn ft fm'  RC=256
DMSSTT062E INVALID * IN FILEID  RC=20
```

LOADMOD

Use the LOADMOD command to load a MODULE file into storage. The file must be in nonrelocatable format as created by the GENMOD command. The format of the LOADMOD command is:

```
┌─────────────────────────────────────────────────────────────────┐
| LOADMod   |  fn [MODULE [fm]]                                   |
|           |              [ *]]                                  |
└─────────────────────────────────────────────────────────────────┘
```

where:

fn is the filename of the file to be loaded into storage. The filetype must be MODULE.

fm is the filemode of the module to be loaded. If not specified, cr specified as an asterisk, all your disks are searched for the file.

Usage Notes

1. You can use the LOADMOD command when you want to debug a CMS MODULE file. After the file is loaded, you may set address stops or breakpoints before you begin execution with the START command; for example:

```
loadmod prog1
cp adstop 210ae
start
```

2. If a MODULE file was created using the DOS option of the GENMCD command, the CMS/DOS environment must be active when it is loaded. If it was created using the OS option (the default), the CMS/DCS environment must not be active when it is loaded.

3. MODULE files created with the ALL option, or with SYSTEM option and loaded into the transient area, may be loaded regardless of whether the CMS/DOS environment is active. If the LOADMOD command is called from a program, the loading is also done regardless of whether the CMS/DOS environment is active.

Responses

None.

Messages and Return Codes

```
DMSMOD001E NO FILENAME SPECIFIED  RC=24
DMSMOD002E FILE 'fn ft' NOT FOUND  RC=28
DMSMOD032E INVALID FILETYPE 'ft'  RC=24
DMSMOD070E INVALID PARAMETER 'parameter'  RC=24
DMSMOD104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSMOD109S VIRTUAL STORAGE CAPACITY EXCEEDED  RC=104
DMSMOD114E 'fn  ft fm'  NOT  LOADED;  CMS/DOS ENVIRONMENT  [NOT]  ACTIVE
           RC=40 or RC=-0005
DMSMOD116S LOADER TABLE OVERFLOW  RC=104
DMSSTT048E INVALID MODE 'mode'  RC=24
```

MACLIB

Use the MACLIB command to create and modify CMS macro libraries. The format of the MACLIB command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
| MAClib |  /  (GEN)                                \                        |
|        | (  < ADD >  libname fn1[fn2...]           )                       |
|        |  |  (REP)                                 |                       |
|        |  |                                        |                       |
|        | <  DEL  libname  membername1[membername2...]>                     |
|        |  |                                        |                       |
|        | (  COMP libname                           )                       |
|        |  |                                        |                       |
|        |  \ MAP  libname [(options...[)]]         /                        |
|        |                                                                   |
|        |                    options:                                       |
|        |                    ┌      ┐                                       |
|        |                    |TERM  |                                       |
|        |                    |DISK  |                                       |
|        |                    |PRINT |                                       |
|        |                    └      ┘                                       |
└──────────────────────────────────────────────────────────────────────────┘
```

where:

GEN generates a CMS macro library.

ADD adds members to an existing macro library. No checking is done for duplicate names, entry points, or CSECTS.

REP replaces existing members in a macro library.

DEL deletes members from a macro library. If more than one member exists with the same name, only the first entry is deleted.

COMP compacts a macro library.

MAP lists certain information about the members in a macro library. Available information includes member name, size, and location relative to the beginning of the library.

libname is the filename of a macro library. If the file already exists, it must have a filetype of MACLIB; if it is being created, it is given a filetype of MACLIB.

fn1 [fn2...]
are the names of the macro definition files to be used. A macro definition file must reside on a CMS disk and its filetype must be either MACRO or COPY. Each file may contain one or more macros and must contain fixed-length, 80-character records.

membername1[membername2...]
are the names of the macros that exist in a macro library.

MAP Options: The following options specify where the output of the MAP function is sent. Only one option may be specified. If more than one option is specified, only the first one given is used.

TERM displays the MAP output at the terminal.

DISK writes the MAP output on a CMS disk with the file identifier of "libname MAP A1". If a file with that name already exists, the old file is erased. If no option is specified, DISK is the default.

PRINT writes the file "libname MAP A1" to your A-disk and spools a copy to the virtual printer.

Usage Notes

1. When a MACRO file is added to a MACLIB, the membername is taken from the macro prototype statement. If there is more than one macro definition in the file, each macro is written into a separate MACLIB member.

   If the filetype is COPY and the file contains more than one macro, each macro must be preceded by a control statement of the following format:

   ```
   *COPY membername
   ```

   The name on the control statement is the name of the macro when it is placed in the macro library. If there is only one macro in the COPY file and it is not preceded by a COPY control statement, its name (in the macro library) is the same as the filename of the COPY file. If there are several macro definitions in a COPY file and the first one is not preceded by a COPY control statement, the entire file is treated as one macro.

2. If any MACRO file contains invalid records between members, the MACLIB command displays an error message and terminates. Any members read before the invalid card is encountered are already in the MACLIB. The MACLIB command ignores CATAL.S, END, and /* records when it reads MACRO files created by the ESERV program.

3. If you want a macro library searched during an assembly or compilation, you must identify it using the GLOBAL command before you begin compiling.

4. The MACLIBs distributed with the CMS system are: CMSLIB, OSMACRC, OSMACRO1, TSOMAC, and DOSMACRO.

5. The TERM or PRINT options will erase the old MAP file, if one exists.

Responses

When you enter the MACLIB MAP command with the TERM option, the names of the library members, their sizes, and their locations in the library are displayed.

```
MACRO INDEX SIZE
name   lcc  size
  .     .    .
  .     .    .
  .     .    .
```

Other Messages and Return Codes

```
  DMSLBM001E NO FILENAME SPECIFIED  RC=24
  DMSLBM002E FILE 'fn ft' NOT FOUND RC=28
  DMSLBM002W FILE 'fn ft [fm]' NOT FOUND  RC=4
  DMSLBM003E INVALID OPTION 'option'  RC=24
  DMSLBM013W MEMBER 'name' NOT FOUND IN LIBRARY 'fn ft fm'  RC=4
  DMSLBM014E INVALID FUNCTION 'function'  RC=24
  DMSLBM037E DISK 'mode' IS READ/ONLY  RC=36
  DMSLBM046E NO LIBRARY NAME SPECIFIED  RC=24
  DMSLBM047E NO FUNCTION SPECIFIED  RC=24
  DMSLBM056E FILE 'fn ft fm' CONTAINS INVALID RECORD FORMATS  RC=32
| DMSLBM069E DISK 'mode' NOT ACCESSED  RC=36
  DMSLBM070E INVALID PARAMETER 'parameter'  RC=24
  DMSLBM104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
  DMSLBM105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
  DMSLBM109S VIRTUAL STORAGE CAPACITY EXCEEDED  RC=104
  DMSLBM157S MACLIB   LIMIT   EXCEEDED[,   LAST   MEMEER   NAME   ADDED   WAS
             'membername'] RC=88
  DMSLBM167S PREVIOUS MACLIB FUNCTION NOT FINISHED  RC=88
  DMSLBM213W LIBRARY 'fn ft fm' NOT CREATED  RC=4
  DMSLBM907T I/O ERROR ON FILE 'fn ft fm'  RC=256
```

MODMAP

Use the MODMAP command to display the load map associated with the specified MODULE file. The format of the MODMAP command is:

MODmap	fn

where:

fn is the filename of the MODULE file whose load map is to be displayed. The filetype of the file must be MODULE; all of your accessed disks are searched for the specified file.

Usage Note

You cannot issue a MODMAP command for modules that are CMS transient area modules or that have been created with the NOMAP option of the GENMOD command.

Responses

The load map associated with the file is displayed at the terminal, in the format:

```
     name      location
       .           .
       .           .
       .           .
```

Error Messages and Return Codes

```
DMSMDP001E NO FILENAME SPECIFIED  RC=24
DMSMDP002E FILE 'fn ft' NOT FOUND  RC=28
DMSMDP018E NO LOAD MAP AVAILABLE  RC=40
DMSMDP070E INVALID PARAMETER 'parameter'  RC=24
```

MOVEFILE

Use the MOVEFILE command to move data from any device supported by VM/370 to any other device supported by VM/370. The format of the MOVEFILE command is:

```
┌──────────────────────────────────────────────────────────────────┐
│            │ ┌             ┌          ┐ ┐                         │
│            │ │inddname     │outddname│ │                         │
│ MOVEfile   │ │             │          │ │   [(PDS[)]]            │
│            │ │INMOVE       │OUTMOVE   │ │                         │
│            │ └             └          ┘ ┘                         │
└──────────────────────────────────────────────────────────────────┘
```

where:

inddname is the ddname representing the input file definition. If ddname is not specified, the default input ddname, INMOVE, is used.

outddname is the ddname representing the output file definition. If ddname is not specified, the default output ddname, OUTMOVE, is used.

Option:

PDS moves each of the members of the CMS MACLIB or TXTLIB or of an OS partitioned data set into a separate CMS disk file, with a filename equal to the member name and a filetype equal to the filetype of the output file definition.

Usage Notes

1. Use the FILEDEF command to provide file definitions for the ddnames used in the MOVEFILE command. If you use the ddnames INMOVE and OUTMOVE on the FILEDEF commands, then you need not specify them on the MOVEFILE command line. For example:

   ```
   filedef inmove disk sys1 maclib b (member stow
   filedef outmove disk stow macro
   movefile
   ```

   copies the member STOW from the OS partitioned data set SYS1.MACLIB into the CMS file STOW MACRO.

   If you enter:

   ```
   filedef indd reader
   filedef outdd printer
   movefile indd outdd
   ```

   a file is moved from your virtual card reader to your virtual printer.

2. To copy an entire OS partitioned data set into individual CMS files, ycu could enter:

   ```
   filedef test2 disk sys1 maclib b
   filedef macro disk
   movefile test2 macro (pds
   ```

   These commands copy members from the OS partitioned data set SYS1.MACLIB or the CMS file SYS1 MACLIB into separate files, each with a filename equal to the membername and a filetype of MACRC. Note that the output ddname was not specified in full, so that CMS assigned the default file definition (FILE ddname).

3. You cannot copy VSAM data sets with the MOVEFILE command.

4. The MOVEFILE command does not support data containing spanned records. Use of spanned records results in the error message DMSSOP036E and an error code of 7.

5. To copy an entire partitioned data set into another partitioned data set, use the COPYFILE command. If an attempt is made to use the MOVEFILE command without the PDS option for a partitioned data set, only the first member is copied and an end-of-file condition results. The resultant output file will contain all input records, including the header, until the end of the first member.

Default Device Attributes

If a record format (RECFM), blocksize (BLOCK), and logical record length (LRECL) are specified on the FILEDEF command, these values are used in the data control block (DCB) defining the characteristics of the move operation. If the FILEDEF was issued without a record format or blocksize specified, these values are determined according to the defaults listed in Figure 17. If the blocksize was not specified, the default blocksize is used. If the logical record length was not specified, the default logical record length is determined as follows: for an F or U record format, the logical record length equals the blocksize; for a V record format, the logical record length equals the blocksize minus 4.

	Input ddname		Output ddname	
Device	RECFM	Blocksize	RECFM	Blocksize
Card Reader	F	80	NA[2]	NA[2]
Card Punch	NA[2]	NA[2]	F	80
Printer	NA[2]	NA[2]	U	132
Terminal	U	130	U	130
Tape[1]	U	3600	RECFM of input ddname	Blocksize of input ddname
Disk file	RECFM of file	Blocksize of file	RECFM of input ddname	Blocksize of input ddname
Dummy	NA[2]	NA[2]	RECFM of input ddname	Blocksize of input ddname

[1] If the default record format and blocksize are used in a tape-to-tape move operation and an input record is greater than 3600 bytes, it is truncated to 3600 bytes on the output tape.

[2] Not applicable.

Figure 17. Default Device Attributes for MOVEFILE Command

Responses

DMSMVE225I PDS MEMBER 'membername' MOVED

The specified member of an OS partitioned data set was moved successfully to a CMS file. This response is issued for each member moved when you use the PDS option.

DMSMVE226I END OF PDS MOVE

The last member of the partitioned data set was moved successfully to a CMS file.

DMSMVE706I TERM INPUT -- TYPE NULL LINE FOR END OF DATA

The input ddname in the MOVEFILE specified a device type of terminal. This message requests the input data; a null line terminates input.

DMSMVE708I DISK FILE 'FILE ddname A1' ASSUMED FOR DDNAME 'ddname'

No file definition is in effect for a ddname specified on the MOVEFILE command. The MOVEFILE issues the default FILEDEF command:

```
FILEDEF ddname DISK FILE ddname A1
```

If file ddname does not exist for the input file, MOVEFILE terminates processing.

Other Messages and Return Codes

```
DMSMVE002E FILE 'fn ft fm' NOT FOUND RC=28
DMSMVE003E INVALID OPTION 'option'  RC=24
DMSMVE037E OUTPUT DISK 'mode' IS READ/ONLY  RC=36
DMSMVE041E INPUT AND OUTPUT FILES ARE THE SAME  RC=40
DMSMVE069E OUTPUT DISK 'mode' IS NOT ACCESSED RC=36
DMSMVE070E INVALID PARAMETER 'parameter'  RC=24
DMSMVE073E UNABLE TO OPEN FILE ddname  RC=28
DMSMVE075E DEVICE 'device name' ILLEGAL FOR {INPUT|OUTPUT}  RC=40
DMSMVE086E INVALID DDNAME 'ddname'  RC=24
DMSMVE127S UNSUPPORTED DEVICE FOR ddname  RC=100
DMSMVE128S I/O ERROR ON  INPUT AFTER READING nnnn  RECORDS:  INPUT ERROR
           code ON ddname  RC=100
DMSMVE129S I/O ERROR ON OUTPUT WRITING  RECORD NUMBER nnnn: OUTPUT ERROR
           code ON ddname  RC=100
DMSMVE130S BLOCKSIZE ON V FORMAT FILE ddname IS LESS THAN 8  RC=88
```

OPTION

Use the OPTION command to change any or all of the options in effect for the DOS/VS COBOL compiler in CMS/DOS. The format of the OPTION command is:

```
  ┌──────────────────────────────────────────────────────────────────────┐
  | OPTION | [options...]                                                |
  |        |                                                             |
  |        |       options:                                              |
  |        |         ┌       ┐ ┌       ┐ ┌       ┐ ┌        ┐ ┌      ┐   |
  |        |         |DUMP   | |DECK   | |LIST   | |LISTX   | |SYM   |   |
  |        |         |NODUMP | |NODECK | |NOLIST | |NOLISTX | |NOSYM |   |
  |        |         └       ┘ └       ┘ └       ┘ └        ┘ └      ┘   |
| |        |            ┌       ┐ ┌       ┐ ┌   ┐ ┌      ┐               |
| |        |            |XREF   | |ERRS   | |48C| |TERM  |               |
| |        |            |NOXREF | |NOERRS | |60C| |NOTERM|               |
| |        |            └       ┘ └       ┘ └   ┘ └      ┘               |
  └──────────────────────────────────────────────────────────────────────┘
```

Options: If an invalid option is specified on the command line, an error message is issued for that option; all other valid options are accepted. Only those options specified are altered, and all other options remain unchanged.

DUMP dumps the registers and the virtual partition on the virtual SYSLST device in the case of abnormal program end.

NODUMP suppresses the DUMP option.

DECK punches the resulting object module on the virtual SYSPCH device. If you do not issue an ASSGN command for the logical unit SYSPCH before invoking the compiler, the text deck is written to your CMS A-disk.

NODECK suppresses the DECK option.

LIST writes the output listing of the source module on the SYSLST device.

NOLIST suppresses the LIST option. This option overrides the XREF option as it does in DOS/VS.

LISTX produces a procedure division map on the SYSLST device.

NOLISTX suppresses the LISTX option.

SYM prints a Data Division map on SYSLST.

NOSYM suppresses the SYM option.

XREF writes the output symbolic cross-reference list on SYSLST.

NOXREF suppresses the XREF option.

ERRS writes an output listing of all errors in the source program on SYSLST.

NOERRS suppresses the ERRS option.

48C Uses the 48-character set.

60C Uses the 60-character set.

TERM	Writes all compiler messages to the user's terminal.
NOTERM	Suppresses the TERM option.

Usage Notes

1. If you enter the OPTION command with no options, all options are reset to their default values, that is, the default settings that are in effect when you enter the CMS/DOS environment. CMS/DOS defaults are not necessarily the same as the defaults generated on the DOS/VSE system being used and do not include additional options that are available with some DOS compilers.

2. The OPTION command has no effect on the DOS/VSE PL/I compiler nor on any of the OS language compilers in CMS.

Responses

None. To display a list of options currently in effect, use the QUERY command with the OPTION operand.

Error Messages and Return Codes

```
DMSOPT070E INVALID PARAMETER 'parameter'  RC=24
DMSOPT099E CMS/DOS ENVIRONMENT NOT ACTIVE  RC=40
```

PRINT

Use the PRINT command to print a CMS file on the spooled virtual printer. The format of the PRINT command is:

```
┌──────────────────────────────────────────────────────────────────────┐
|           |                                                          |
|  PRint    |  fn ft |fm|  [(options...[)]]                            |
|           |        |* |                                              |
|           |        └  ┘                                              |
|           |                                                          |
|           |          options: |CC  |              |LINECOUN {nn} |   |
|           |                   |NOCC|  [UPCASE]    |         {55} |   |
|           |                   └    ┘              └              ┘   |
|           |                   ┌                     ┐                |
|           |                   |MEMBER { *          }|                |
|           |                   |       { membername }|  [HEX]         |
|           |                   └                     ┘                |
└──────────────────────────────────────────────────────────────────────┘
```

where:

fn is the filename of the file to be printed.

ft is the filetype of the file to be printed.

fm is the filemode of the file to be printed. If this field is specified as an asterisk (*), the standard order of search is followed and the first file found with the given filename and filetype is printed. If fm is not specified, the A-disk and its extensions are searched.

Options:

CC interprets the first character of each record as a carriage control character. If the filetype is LISTING, the CC option is assumed. If CC is in effect, the PRINT command does not perform page ejects nor count the number of lines per page; these functions are controlled by the carriage control characters in the file. The LINECOUN option has no effect if CC is in effect.

NOCC does not interpret the first character of each record as a carriage control character. In this case, the PRINT command ejects a new page and prints a heading after the number of lines specified by LINECOUN are printed. If NOCC is specified, it is in effect even if CC was specified previously or if the filetype is LISTING.

UPCASE
UP translates the lowercase letters in the file to uppercase for printing.

MEMBER { * }
MEM { membername }

prints the members of macro or text libraries. This option may be specified if the file is a simulated partitioned data set (filetype MACLIB or TXTLIB). If an asterisk (*) is entered, all individual members of that library are printed. If a membername is specified, only that member is printed.

HEX prints the file in graphic hexadecimal format. If HEX is specified, the options CC and UPCASE are ignored, even if specified, and even if the filetype is LISTING.

```
LINECOUN { nn }
LI       { 55 }
```

allows you to set the number of lines to be printed on each page. nn can be any decimal number from 0 through 99. If a number is not specified, the default value is 55. If nn is set to zero, the effect is that of an infinite line count and page ejection does not occur. This option has no effect if the CC option is also specified.

Usage Notes

1. The file may contain carriage control characters and may have either fixed- or variable-length records, but no record may exceed 132 characters for a 1403, 3203, or 3289 Model 4 printer or 150 characters for a 3211 printer. There are two exceptions:

   - If the CC option is in effect, the record length can be one character longer (133 or 151) to allow for the carriage control character.

   - If the HEX option is in effect, a record of any length can be printed, up to the CMS file system maximum of 65,535 bytes.

2. If you want the first character of each line to be interpreted as a carriage control character, you must use the CC option. When you use the CC option for files that do not contain carriage control characters, the first character of each line is stripped off. An attempt is made to interpret the first character for carriage control purposes, and the results are unpredictable.

   Files with a filetype of UPDLOG (produced by the UPDATE command) must be printed with the CC option.

3. One spool printer file is produced for each PRINT command; for example:

   ```
   print mylib maclib (member get
   ```

   prints the member GET from the file MYLIB MACLIB. If you want to print a number of files as a single file (so that you do not get output separator pages, for example), use the CP command SPOOL to spool your virtual printer with the CONT option.

4. The PRINT command has its own forms control buffer load. The format of the FCB macro used is:

   ```
   FCB NNNN, 6,66, (1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10,
                    11,64,12,65,9)
   ```

   This FCB macro is always loaded by the PRINT command and must be taken into account when the CC option is used.

Responses

None. The CMS ready message indicates the command completed without error (that is, the file is written to the spooled printer). The file is now under the control of CP spooling functions. If a CP SPOOL command option such as HOLD or COPY is in effect, you may receive a message from CP.

Other Messages and Return Codes

```
DMSPRT002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSPRT003E INVALID OPTION 'option'  RC=24
DMSPRT008E DEVICE 'cuu' {INVALID OR NONEXISTENT|UNSUPPORTED DEVICE TYPE}
           RC=36
DMSPRT013E MEMBER 'name' NOT FOUND IN LIBRARY  RC=32
DMSPRT029E INVALID PARAMETER  'parameter' IN  THE OPTION  'option' FIELD
           RC=24
DMSPRT033E FILE 'fn ft fm' IS NOT A LIBRARY  RC=32
DMSPRT039E NO ENTRIES IN LIBRARY 'fn ft fm'  RC=32
DMSPRT044E RECORD LENGTH EXCEEDS ALLOWABLE MAXIMUM  RC=32
DMSPRT048E INVALID MODE 'mode'  RC=24
DMSPRT054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSPRT062E INVALID * IN FILEID  RC=20
DMSPRT070E INVALID PARAMETER 'parameter'  RC=24
DMSPRT104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSPRT123S ERROR PRINTING FILE 'fn ft fm'  RC= 100
```

PSERV

Use the PSERV command in CMS/DOS to copy, display, print, or punch a procedure from the DOS/VSE procedure library. The format of the PSERV command is:

```
┌──────────────────────────────────────────────────────────────────────┐
│        │              ┌    ┐                                         │
│PSERV   │ procedure    │ ft │ [ (options... [) ]]                     │
│        │              │PROC│                                         │
│        │              └    ┘        options:                         │
│        │                                 [DISK]   [PRINT]            │
│        │                                                             │
│        │                                 [PUNCH]  [TERM]             │
└──────────────────────────────────────────────────────────────────────┘
```

where:

procedure specifies the name of the procedure in the DOS procedure library that you want to copy, print, punch, or display.

ft specifies the filetype of the file to be created on your A-disk. ft defaults to PROC if a filetype is not specified; the filename is always the same as the procedure name.

Options: You may enter as many options as you wish, depending on the functions you want to perform.

DISK copies the procedure to a CMS file. If no options are specified, DISK is the default.

PRINT spools a copy of the procedure to the virtual printer.

PUNCH spools a copy of the procedure to the virtual punch.

TERM displays the procedure on your terminal.

Usage Notes

1. You cannot execute DOS/VSE procedures in CMS/DOS. You can use the PSERV command to copy an existing DOS/VSE procedure onto a CMS disk, use the CMS Editor to change or add DOS/VSE job control statements to it, and then spool it to the reader of a DOS/VSE virtual machine for execution.

2. The PSERV command ignores current assignments of logical units, and directs output according to the option list.

Responses

When you issue the TERM option, the procedure is displayed at your terminal.

Error Messages and Return Codes

```
DMSPRV003E INVALID OPTION 'option'  RC=24
DMSPRV004E PROCEDURE 'procedure' NOT FOUND  RC=28
DMSPRV006E NO READ/WRITE 'A' DISK ACCESSED  RC=36
DMSPRV037E DISK 'A' IS READ/ONLY  RC=36
DMSPRV070E INVALID PARAMETER 'parameter'  RC=24
DMSPRV097E NO 'SYSRES' VOLUME ACTIVE  RC=36
DMSPRV098E NO PROCEDURE NAME SPECIFIED  RC=24
DMSPRV099E CMS/DOS ENVIRONMENT NOT ACTIVE  RC=40
DMSPRV105S ERROR 'nn' WRITING FILE 'fn ft fm' TO DISK  RC=100
DMSPRV113S DISK (cuu) NOT ATTACHED  RC=100
DMSPRV411S INPUT ERROR CODE 'nn' ON 'SYSRES'  RC=100
```

PUNCH

Use the PUNCH command to punch a CMS disk file to your virtual card punch. The format of the PUNCH command is:

```
┌──────────────────────────────────────────────────────────────────────┐
│              │          ┌  ┐                                          │
│ PUnch        │   fn ft  │fm│  [(options...[)]]     options:            │
│              │          │* │                                          │
│              │          └  ┘                                          │
│              │                                      ┌        ┐        │
│              │                                      │HEADER  │        │
│              │                                      │NOHEADER│        │
│              │                                      └        ┘        │
│              │                                      ┌                 ┐│
│              │                                      │MEMBER {*         }││
│              │                                      │       {membername}││
│              │                                      └                 ┘│
│              │                                                        │
└──────────────────────────────────────────────────────────────────────┘
```

where:

fn is the filename of the file to be punched. This field must be specified.

ft is the filetype of the file to be punched. This field must be specified.

fm is the filemode of the file to be punched. If you specify it as an asterisk (*), the standard order of search is followed and the first file found with the specified filename and filetype is punched. If fm is not specified, your A-disk and its extensions are searched.

Options:

HEADER
H
inserts a control card in front of the punched output. This control card indicates the filename and filetype fcr a subsequent READCARD command to restore the file to a disk. The control card format is shown in Figure 18.

NOHEADER
NOH
does not punch a header control card.

MEMBER {*
MEM {membername}
punches members of MACLIBs or TXTLIBs. If an asterisk (*) is entered, all individual members of that macro or text library are punched. If membername is specified, only that member is punched. If the filetype is MACLIB and the MEMBER membername option is specified, the header contains MEMBER as the filetype. If the filetype is TXTLIB and the MEMBER membername option is specified, the header card contains TEXT as the filetype.

Column	Number of Characters	Contents	Meaning
1	1	:	Identifies card as a control card.
2-5	4	READ	Identifies card as a READ control card.
6-7	2	blank	
8-15	8	fname	Filename of the file punched.
16	1	blank	
17-24	8	ftype	Filetype of the file punched.
25	1	blank	
26-27	2	fmode	Filemode of the file punched.
28	1	blank	
29-34	6	volid	Label of the disk from which the file was read.
35	1	blank	
36-43	8	mm/dd/yy	The date that the file was last written.
44-45	2	blank	
46-50	5	hh:mm	The time of day that the file was written to disk.
51-80	30	blank	

Figure 18. Header Card Format

Usage Notes

1. You can punch fixed- or variable-length records with the PUNCH command, as long as no record exceeds 80 characters. Records with less than 80 characters are right-padded with blanks. Records longer than 80 characters are rejected.

2. If you punch a MACLIB or TXTLIB file specifying the MEMBER * option, a read control card is placed in front of each library member. If you punch a library without specifying the MEMBER * option, only one read control card is placed at the front of the deck.

3. One spool punch file is produced for each PUNCH command; for example:

   ```
   punch compute assemble (noh
   ```

   punches the file COMPUTE ASSEMBLE, without inserting a header card. To transmit multiple CMS files as a single punch file, use the CP SPOOL command to spool the punch with the CONT option.

Responses

None. The CMS ready message indicates that the command completed without error (the file was successfully spooled); the file is now under control of CP spooling functions. You may receive a message from CP indicating that the file is being spooled to a particular user's virtual card reader.

Other Messages and Return Codes

```
DMSPUN002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSPUN003E INVALID OPTION 'option'  RC=24
DMSPUN008E DEVICE 'cuu' {INVALID OR NONEXISTENT|UNSUPPORTED DEVICE TYPE}
           RC=36
DMSPUN013E MEMBER 'name' NOT FOUND IN LIBRARY  RC=32
DMSPUN033E FILE 'fn ft fm' IS NOT A LIBRARY  RC=32
DMSPUN039E NO ENTRIES IN LIBRARY 'fn ft fm'  RC=32
DMSPUN044E RECORD LENGTH EXCEEDS ALLOWABLE MAXIMUM  RC=32
DMSPUN054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSPUN062E INVALID * IN FILEID  RC=20
DMSPUN104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSPUN118S ERROR PUNCHING FILE 'fn ft fm'  RC=100
```

QUERY

Use the QUERY command to gather information about your CMS virtual machine. You can determine:

- The state of virtual machine characteristics that are controlled by the CMS SET command
- File definitions (set with the FILEDEF and DLBL commands) that are in effect
- The status of accessed disks
- The status of CMS/DOS functions

The format of the QUERY command is:

```
| Query      |   BLIP
|            |   RDYMSG
|            |   LDRTBLS
|            |   RELPAGE
|            |   IMPCP
|            |   IMPEX
|            |   ABBREV
|            |   REDTYPE
|            |   PROTECT
|            |   INPUT
|            |   OUTPUT
|            |   SYSNAMES
|            |   SEARCH
|            |
|            |   DISK     [mode]
|            |            [ *  ]
|            |
|            |            SYSTEM
|            |   SYNONYM  USER
|            |            ALL
|            |
|            |   FILEDEF
|            |   LABELDEF
|            |   MACLIB
|            |   TXTLIB
|            |   LIBRARY
|            |
|            |CMS/DOS Functions:
|            |
|            |   DLBL
|            |   DOS
|            |   DOSLIB
|            |   DOSPART
|            |   DOSLNCNT
|            |   OPTION
|            |   UPSI
```

Operands for Functions that Can Be Controlled Via the SET Command:

BLIP displays the BLIP character(s).

Response: BLIP = { xxxxxxxx | OFF }

RDYMSG displays the RDYMSG message of the CMS Ready format.

Response: RDYMSG = $\left\{\begin{matrix}\text{LMSG}\\\text{SMSG}\end{matrix}\right\}$

where:

LMSG is the standard CMS Ready message:

```
R; T = 0.12/0.33 17:06:20
```

SMSG is the shortened CMS Ready message:

```
R;
```

LDRTBLS displays the number of loader tables.

Response: LDRTBLS = nn

RELPAGE indicates whether pages of storage are to be released or retained after certain commands complete execution.

Response: RELPAGE = $\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$

where:

ON releases pages.
OFF retains pages.

IMPCP displays the status of implied CP command indicator.

Response: IMPCP = $\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$

where:

ON indicates that CP commands can be entered from the CMS environment.

OFF indicates that you must use the CP command or the #CP function to enter CP commands from the CMS environment.

IMPEX displays status of implied EXEC indicator.

Response: IMPEX = $\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$

where:

ON indicates that EXEC files can be executed by entering the filename of the file.

OFF indicates that the EXEC command must be explicitly entered to execute EXEC files.

ABBREV displays the status of the minimum truncation indicator.

Response: ABBREV = $\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$

where:

ON indicates that truncations are accepted for CMS commands.
OFF indicates that truncations are not accepted.

REDTYPE displays the status of the REDTYPE indicator.

Response: REDTYPE = $\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$

where:

ON types CMS error messages in red, for certain terminals equipped with the appropriate terminal feature and a two-color ribbon. Supported terminals are described in the VM/370 Terminal User's Guide.

OFF does not type CMS error messages in red.

PROTECT displays the status of CMS nucleus protection.

Response: PROTECT = $\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$

where:

ON means CMS nucleus protection is in effect.
OFF means CMS nucleus protection is not in effect.

INPUT displays the contents of any input translate table in effect.

```
Response:  INPUT  a1  xx1
                  .   .
                  .   .
                  .   .
                  an  xxn
```

If you do not have an input translate table in effect, the response is:

```
NO USER DEFINED INPUT TRANSLATE TABLE IN USE
```

OUTPUT displays the contents of any output translate table in effect.

```
Response:  OUTPUT  xx1  a1
                   .    .
                   .    .
                   .    .
                   xxn  an
```

If you do not have an output translate table defined, the response is:

```
NO USER DEFINED OUTPUT TRANSLATE TABLE IN USE
```

SYSNAMES displays the names of the saved system currently being used by your virtual machine.

```
Response:  SYSNAMES:  CMSSEG   CMSVSAM  CMSAMS   CMSDOS
            ENTRIES:  entry... entry... entry... entry...
```

__where__:

SYSNAMES are the standard names that identify the discontiguous saved systems.

ENTRIES are the names of the saved systems being used, if the saved systems exist.

__Operands__ __for__ __CMS__ __Disk__ __Status__ __Functions__:

SEARCH displays the search order of all disks currently accessed.

```
                                      ┌     ┐
Response: label   cuu   mode ┌R/O┐ |-OS  |
                             └R/W┘ |-DOS|
                                      └     ┘
            .       .      .     .
            .       .      .     .
            .       .      .     .
```

__where__:

label is the label assigned to the disk when it was formatted; or, if it is an OS or DOS disk, the volume label.

cuu is the virtual device address.

mode is the filemode letter assigned to the disk when it was accessed.

```
{R/O} indicates whether read/write or read-only is the status
{R/W} of the disk.
```

```
┌    ┐
|OS  |  indicates an OS or DOS disk.
|DOS|
└    ┘
```

DISK mode displays the status of the single disk represented by "mode".

__Response__:

```
LABEL CUU M STAT  CYL TYPE BLKSIZE  FILES BLKS USED-(%) BLKS LEFT BLK TOTAL
label cuu m {R/O} cyl type blksize   nnnn      nnnn-nn        nnnn     nnnnn
            {R/W}
```

If the disk is an OS or DOS disk, the response is:

```
LABEL CUU M STAT   CYL  TYPE BLKSIZE  FILES BLKS USED-(%) BLKS LEFT BLK TOTAL
label cuu m {R/O} {cyl} type           { OS}
            {R/W} {FBA}                {DOS}
```

__where__:

label is the label assigned to the disk when it was formatted; or, if it is an OS or DOS disk, the volume label.

cuu is the virtual device address.

m is the access mode letter.

$\left\{\begin{matrix}\text{R/O}\\ \text{R/W}\end{matrix}\right\}$ STAT indicates whether read/write or read-only is the status of the disk.

cyl is the number of cylinders available on the disk. For an FB-512 device, this field contains the notation 'FBA' rather than the number of cylinders.

type is the device type of the disk.

blksize is the CMS disk block size when the minidisk was formatted.

nnnn FILES is the number of CMS files on the disk.

nnnn BLKS USED indicates the number of CMS disk blocks in use.

nn % indicates the percentage of blocks in use.

nnnn BLKS LEFT indicates the number of disk blocks left. This is a high approximation because control blocks are included.

nnnnn BLK TOTAL indicates the total number of disk blocks.

```
{OS }  indicates an OS or DOS disk.
{DOS}
```

If the disk with the specified mode is not accessed, the response is:

```
DISK 'mode' NOT ACCESSED
```

DISK * displays the status of all CMS disks.

__Response__: Is the same as for QUERY DISK mode; one line is displayed for each accessed disk.

__Other Functions__:

SYNONYM SYSTEM
displays the CMS system synonyms in effect.

__Response__:

```
SYSTEM   SHORTEST
COMMAND  FORM
-------  ------------------
command  minimum truncation
   .           .
   .           .
   .           .
```

If no system synonyms are in effect, the following message is displayed at the terminal:

```
NO SYSTEM SYNONYMS IN EFFECT
```

SYNONYM USER
displays user synonyms in effect.

__Response__:

```
SYSTEM  USER     SHORTEST
COMMAND SYNONYM  FORM (IF ANY)
------- -------  ------------------
command synonym  minimum truncation
   .       .               .
   .       .               .
   .       .               .
```

If no user synonyms are in effect, the following message is displayed at the terminal:

```
NO USER SYNONYMS IN EFFECT
```

SYNONYM ALL
displays all synonyms in effect.

__Response__: The response to the command QUERY SYNONYM SYSTEM is followed by the response to QUERY SYNONYM USER.

FILEDEF displays all file definitions in effect.

```
Response: ddname device [fn [ft]]
            .      .      .   .
            .      .      .   .
            .      .      .   .
```

If no file definitions are in effect, the following message is displayed at the terminal:

```
NO USER DEFINED FILEDEF'S IN EFFECT
```

LABELDEF displays all label definitions in effect.

```
Response: ddname volid fseq volseq genn genv crdte exdte fid
            .      .     .     .     .    .     .     .    .
            .      .     .     .     .    .     .     .    .
            .      .     .     .     .    .     .     .    .
```

Only fields you have explicitly specified are displayed. Defaulted fields are not displayed. If no label definitions are in effect, the following message is displayed at the terminal:

```
NO USER DEFINED LABELDEF'S IN EFFECT
```

MACLIB displays the names of all files, with a filetype of MACLIB, that are to be searched for macro definitions (that is, all MACLIBs specified on the last GLOBAL MACLIB command, if any).

```
Response: MACLIB = libname...
```

If no macro libraries are to be searched for macro definitions, the response is:

```
MACLIB = NONE
```

TXTLIB displays the names of all files, with a filetype of TXTLIB, that are to be searched for unresolved references (that is, all TXTLIBs specified on the last GLOBAL TXTLIB command, if any).

```
Response: TXTLIB = libname...
```

If no TXTLIBs are to be searched for unresolved references, the following message is displayed at the terminal:

```
TXTLIB = NONE
```

LIBRARY displays the names of all library files with filetypes of MACLIB, TXTLIB, and DOSLIB that are to be searched.

```
Response: MACLIB = {libname...}
                   {NONE      }

          TXTLIB = {libname...}
                   {NONE      }

          DOSLIB = {libname...}
                   {NONE      }
```

CMS/DOS Functions:

DLBL in order to display the contents of the current data set definitions, it is necessary only to enter:

```
DLBL  or  QUERY DLBL
```

Entering the command yields the following information:

DDNAME the DOS filename or OS ddname.

MODE — the CMS disk mode identifying the disk on which the data set resides.

LOGUNIT — the DOS logical unit specification (SYSxxx). This operand will be blank for a data set defined while in CMS/OS environment; that is, the SET DOS ON command had not been issued at DLBL definition time.

TYPE — indicates the type of data set defined. This field may only have the values SEQ (sequential) and VSAM.

CATALOG — indicates the ddname of the VSAM catalog to be searched for the specified data set. This field will be blank for sequential (SEQ) dataset definitions.

EXT — specifies the number of extents defined for the data set. The actual extents may be displayed by entering either the DLBL (EXTENT) or the QUERY DLBL EXTENT command. This field will be blank if no extents are active for a VSAM data set or if the data set is sequential (SEQ).

VOL — specifies the number (if greater than one) of volumes on which the VSAM data set resides. The actual volumes may be displayed by entering either the DLBL (MULT) or the QUERY DLBL MULT commands. This field will be blank if the VSAM data set resides only on one volume or if the data set is sequential (SEQ).

BUFSP — indicates the size of the VSAM buffer space if entered at DLBL definition time. This field will be blank if the dataset is sequential (SEQ).

PERM — indicates whether the DLBL definition was made with the PERM option. The field will contain YES or NO.

DISK — indicates whether the data set resided on a CMS or DOS/OS disk at DLBL definition time. The values for this field are DOS and CMS.

DATASET.NAME
for a data set residing on a CMS disk, the CMS filename and filetype are given; for a data set residing on a DOS/OS disk, the data set name (maximum 44 characters) is given. This field will be blank if no DOS/OS data set name is entered at DLBL definition time.

If no DLBL definitions are active, the following message is issued:

```
DMSDLB324I  NO USER DEFINED DLBL'S IN EFFECT
```

DOS — displays whether the CMS/DOS environment is active or not.

Response: DOS = $\left\{\begin{array}{l}\text{ON}\\\text{OFF}\end{array}\right\}$

DOSLIB — displays the names of all files with a filetype of DOSLIB that are to be searched for executable phases (that is, all DOSLIBs specified on the last GLOBAL DOSLIB command, if any).

Response: DOSLIB = $\left\{\begin{array}{l}\text{libname ...}\\ \text{NONE}\end{array}\right\}$

DOSPART displays the current setting of the virtual partition size.

Response: $\left\{\begin{array}{l}\text{nnnnnK}\\ \text{NONE}\end{array}\right\}$

where:

nnnnnK indicates the size of the virtual partition to be used at program execution time.

NONE indicates that CMS determines the virtual partiticn size at program execution time.

DOSLNCNT displays the number of SYSLST lines per page.

Response: DOSLNCNT = nn

where:

nn is an integer from 30 to 99.

OPTION displays the compiler options that are currently in effect.

Response: OPTION = options...

UPSI displays the current setting of the UPSI byte. The eight individual bits are displayed as zeros or ones depending upcn whether the corresponding bit is on or off.

Response: UPSI = nnnnnnnn

Usage Notes

1. You can specify only one QUERY command function at a time. If the implied CP function is in effect and you enter an invalid QUERY command function, you may receive the message DMKCQG045E.

2. If an invalid QUERY command function is specified from an EXEC and the implied CP function is in effect, then the return code is -0003.

3. The DOSPART, OPTION, and UPSI functions are valid only if the CMS/DOS environment is active.

Error Messages and Return Codes

```
DMSQRY005E NO 'option' SPECIFIED  RC=24
DMSQRY014E INVALID FUNCTION 'function'  RC=24
DMSQRY026E INVALID PARAMETER 'parameter' FOR 'function' FUNCTION  RC=24
DMSQRY047E NO FUNCTION SPECIFIED  RC=24
DMSQRY070E INVALID PARAMETER 'parameter'  RC=24
DMSQRY099E CMS/DOS ENVIRONMENT NOT ACTIVE  RC=40
```

READCARD

Use the READCARD command to read data records from your virtual card reader and to create CMS disk files containing the data records. The format of the READCARD command is:

```
┌──────────────────────────────────────────────────────────────────────┐
│ READcard  │  /           ┌    ┐  \                                    │
│           │ |   fn ft    │ fm │   |                                   │
│           │ |            │ A  │   |                                   │
│           │  \           └    ┘  /                                    │
│           │  /     ┌   ┌     ┐┐  \                                    │
│           │ |   *  │ * │ fm  ││   |                                   │
│           │ |      │   │ A   ││   |                                   │
│           │  \     └   └     ┘┘  /                                    │
└──────────────────────────────────────────────────────────────────────┘
```

where:

fn   is the filename you want to assign to the file being read.

ft   is the filetype you want to assign to the file being read.

* [*]   indicates that file identifiers are to be assigned according to READ control cards in the input deck.

fm   is the filemode of the disk onto which the file is to be read. If this field is omitted or specified as an asterisk (*), the A-disk is assumed. Whenever a mode number is specified on the command line, it is used; otherwise, the mode number on the READ control card is used to create the disk file.

Usage Notes

1. Data records read by the READCARD command must be fixed-length records, and may be a minimum of 80 and a maximum of 151 characters long.

2. CMS disk file identifiers are assigned according to READ control cards in the input deck (the PUNCH command header card is a valid READ control card). When you enter the command:

   ```
   readcard *
   ```

   CMS reads the first spool reader file in the queue and if there are READ control cards in the input stream, it names the files as indicated on the control cards.

   If the first card in the deck is not a READ control card, CMS writes a file named READCARD CMSUT1 A1 to contain the data, until a READ control card is encountered or until the end-of-file is reached.

3. If you specify a filename and filetype on the READCARD command, for example:

   ```
   readcard junk file
   ```

   CMS does not check the input stream for READ control cards, but reads the entire spool file onto disk and assigns it the specified filename and filetype.

If there were any READ control cards in the deck, they are nct removed; you must delete them using the CMS Editor if you do nct want them in your file. If the file is too large, you can either increase the size of your virtual storage (using the CP DEFINE command), or use the COPYFILE command to copy all records except the READ control cards (using the FROM and FOR options).

4. To read a file onto a disk other than your A-disk, you can specify the filemode letter when you enter the filename and filetype; for example:

   ```
   readcard junk file c
   ```

   Or, if you want READ control card to determine the filenames and filetypes, you can enter:

   ```
   readcard * * c
   ```

5. When you read a file that has the same filename and filetype as that of an existing file on the same disk, the old file is replaced.

6. If you are preparing real or virtual card decks to send to your own or another user's virtual card reader, you may insert READ control cards to designate filenames, filetypes, and optionally, filemode numbers, to be assigned to the disk file(s).

   A READ control card must begin in column 1 and has the format:

   ```
   :READ filename filetype filemode
   ```

   Each field must be separated by at least one blank; the second character of the filemode field, if specified, must be a valid filemode number (0 through 5). The filemode letter is ignored when this file is read, since the mode letter is determined by specifications on the READCARD command line.

7. To send a real card deck to your own or another user's virtual card reader, you must punch a CP ID card to precede the deck. The ID card has the keyword ID or USERID in column 1, followed by the userid you want to receive the file and optionally, spool file class and name designations; for example:

   ```
   ID CONCARNE CLASS A NAME CHILI PEPPER
   ```

   Each field must be separated from the others by at least one blank.

Responses

When the READCARD * command is issued, control cards encountered in the input card stream are displayed at the terminal (see message DMSRDC702I), to indicate the names assigned to each file.

DMSRDC701I NULL FILE

The spooled card reader contains no records after the control card.

DMSRDC702I :READ filename filetype fn (other information)

A READ control card has been processed; the designated file is being written on disk.

DMSRDC702I READ CONTROL CARD IS MISSING. FOLLOWING ASSUMED:
DMSRDC702I :READ READCARD CMSUT1 A1

The first card in the deck is not a READ control card. Therefore, the file READCARD CMSUT1 A1 is created.

DMSRDC738I RECORD LENGTH IS 'nnn' BYTES

The records being read are not 80 bytes long; this message gives the length.

Other Messages and Return Codes

```
DMSRDC008E DEVICE 'cuu' {INVALID OR NONEXISTENT|UNSUPPORTED DEVICE TYPE}
           RC=36
DMSRDC042E NO FILEID SPECIFIED  RC=24
DMSRDC054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSRDC062E INVALID * IN FILEID  RC=20
DMSRDC105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSRDC124S ERROR READING CARD FILE  RC=100
DMSRDC205W READER EMPTY OR NOT READY  RC=8
```

RELEASE

Use the RELEASE command to free an accessed disk and make the files on it unavailable. The format of the RELEASE command is:

```
┌──────────────────────────────────────────────────────────────┐
│ RELease  │ ⎧ cuu  ⎫ [(DET[)]]                                │
│          │ ⎩ mode ⎭                                          │
└──────────────────────────────────────────────────────────────┘
```

where:

cuu is the virtual device address of the disk that is to be released.

Valid addresses are 001 through 5FF for a virtual machine in basic control mode and 001 through FFF for a virtual machine in extended control mode.

mode is the mode letter at which the disk is currently accessed.

Option:

DET specifies that the disk is to be detached from your virtual machine configuration; CMS calls the CP command DETACH.

Usage Notes

1. If a disk is accessed at more than one mode letter, the RELEASE cuu command releases all modes. If you access a disk specifying the mode letter of an active disk, the first disk is released.

2. You cannot release the system disk (S-disk).

3. When a disk is released, the user file directory is freed from storage and that storage becomes available for other CMS commands and programs. When you release a read/write CMS disk, either with the RELEASE command or implicitly with the FORMAT command, the user file directory is sorted and rewritten on disk; user(s) who may subsequently access the same disk may have a resultant favorable decrease in file search time.

4. When a disk is released, any read-only extensions it may have are not released. The extensions may be referred to by their own mode letters. If a disk is then accessed with the same mode as the original parent disk, the original read-only extensions remain extensions to the new disk at that mode.

5. In CMS/DOS, when you release a disk, any system or programmer logical unit assignments made for the disk are unassigned.

Responses

DASD cuu DETACHED

This is a CP message that is issued when you use the DET option. It indicates that the disk has been detached.

Error Messages and Return Codes

```
DMSARE017E INVALID DEVICE ADDRESS 'cuu'  RC=24
DMSARE028E NO DEVICE SPECIFIED  RC=24
DMSARE048E INVALID MODE 'mode'  RC=24
DMSARE069E DISK {'mode'|'cuu'}  NOT ACCESSED  RC=36
DMSARE070E INVALID PARAMETER 'parameter'  RC=24
```

RENAME

Use the RENAME command to change the fileid of one or more CMS files cn a read/write CMS disk. The format of the RENAME command is:

```
┌──────────────────────────────────────────────────────────────────────┐
│  Rename  │  fileid1 fileid2 [(options...[)]]                         │
│          │                                                           │
│          │                    options:                               │
│          │                                                           │
│          │                  ┌       ┐  ┌        ┐                    │
│          │                  │TYPE   │  │UPDIRT  │                    │
│          │                  │NOTYPE │  │NOUPDIRT│                    │
│          │                  └       ┘  └        ┘                    │
└──────────────────────────────────────────────────────────────────────┘
```

where:

fileid1 is the file identifier of the original file whose name is to be changed. All components of the fileid (filename, filetype, and filemode) must be coded, with either a name cr an asterisk. If an asterisk is coded in any field, any file that satisfies the other qualifications is renamed.

fileid2 is the new file identifier of the file. All components of the file (filename, filetype, and filemode) must be coded, with either a name or an equal sign; if an equal sign (=) is coded, the corresponding file identifier is unchanged. The output filemode can also be specified as an asterisk (*), indicating that the filemode is not changed.

Options:

TYPE
T
displays, at the terminal, the new identifiers of all the files that are renamed. The file identifiers are displayed only when an asterisk (*) is specified for one or more of the file identifiers (fn, ft, or fm) in fileid1.

NOTYPE
NOT
suppresses at the terminal, displaying of the new file identifiers of all files renamed.

UPDIRT
UP
updates the master file directory upon completion of this command.

NOUPDIRT
NOUP
suppresses the updating of the master file directory upon completion of this command. (See Usage Note 3.)

Usage Notes

1. When you code an asterisk (*) in any portion of the input fileid, any or all of the files that satisfy the other qualifiers may be renamed, depending upon how you specify the output fileid. For example:

   ```
   rename * assemble a test file a
   ```

   results in the first ASSEMBLE file found on the A-disk being renamed to TEST FILE. If more than one ASSEMBLE file exists, errcr messages are issued to indicate that they cannot be renamed.

If you code an equal sign (=) in an output fileid in a position corresponding to an asterisk in an input fileid, all files that satisfy the condition are renamed. For example:

```
rename * assemble a = oldasm =
```

renames all files with a filetype of ASSEMBLE to files with a filetype of OLDASM. Current filenames are retained.

2. You cannot use the RENAME command to move a file from one disk to another. You must use the COPYFILE command if you want to change filemode letters.

   You can use the RENAME command to modify filemode numbers, for example,

   ```
   rename * module a1 = = a2
   ```

   changes the filemode number on all MODULE files that have a mode number of 1 to a mode number of 2.

   Note: You can invoke the RENAME command from the terminal, from an EXEC file, or as a function from a program. If RENAME is invoked as a function or from an EXEC file that has the &CONTROL NOMSG option in effect, the message DMSRNM002E FILE 'fn ft fm' NOT FOUND is not issued.

3. Normally, the file directory for a CMS disk is updated whenever you issue a command that affects files on the disk. When you use the NOUPDIRT option of the RENAME command, the file directory is not updated until you issue a command that writes, updates, or deletes any file on the disk, or until you explicitly release the disk (with the RELEASE command).

Responses

newfn newft newfm

The new filename, filetype, and filemode of each file altered is displayed when the TYPE option is specified and an asterisk was specified for at least one of the file identifiers (fn, ft or fm) of the input fileid.

Error Messages and Return Codes

```
DMSRNM002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSRNM003E INVALID OPTION 'option'  RC=24
DMSRNM019E IDENTICAL FILEIDS  RC=24
DMSRNM024E FILE 'fn ft fm' ALREADY EXISTS  RC=28
DMSRNM030E FILE 'fn ft fm' ALREADY ACTIVE  RC=28
DMSRNM037E DISK 'mode(cuu)' IS READ/ONLY  RC=36
DMSRNM048E INVALID FILE MODE 'fm' RC=24
DMSRNM051E INVALID MODE CHANGE  RC=24
DMSRNM054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSRNM062E INVALID * IN OUTPUT FILEID  RC=20
```

RSERV

Use the RSERV command in CMS/DOS to copy, display, print, or punch a DOS/VSE relocatable module from a private or system library. The format of the RSERV command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│        │            ┌      ┐                                             │
│ RSERV  │ mcdname    │ ft   │ [(options...[)]]                            │
│        │            │ TEXT │                                             │
│        │            └      ┘            options:                         │
│        │                                    [DISK]    [PRINT]            │
│        │                                    [PUNCH    [TERM]             │
└──────────────────────────────────────────────────────────────────────────┘
```

where:

modname specifies the name of the module on the DOS/VSE private or system relocatable library. The private library, if any, is searched before the system library.

ft specifies the filetype of the file to be created on your A-disk. ft defaults to TEXT if a filetype is not specified. The filename is always the same as the module name.

Options: You may specify as many options as you wish on the RSERV command, depending on which functions you want to perform.

DISK copies the relocatable module onto your A-disk. If no other options are specified, DISK is the default.

PUNCH punches the relocatable module on the virtual punch.

PRINT prints the relocatable module on the virtual printer.

TERM displays the relocatable module at your terminal.

Usage Notes

1. If you want to copy modules from a private relocatable library, you must issue an ASSGN command for the lcgical unit SYSRLB and identify the library on a DLBL command line using the ddname IJSYSRL.

   To copy modules from the system relocatable library, you must have entered the CMS/DOS environment specifying a mode letter on the SET DOS ON command line.

2. The RSERV command ignores the assignment of logical units, and directs output to the devices specified on the option list.

Responses

If you use the TERM option, the relocatable module is displayed at the terminal.

Messages and Return Codes

```
DMSRRV003E INVALID OPTION 'option'  RC=24
DMSRRV004E MODULE 'module' NOT FOUND  RC=28
DMSRRV006E NO READ/WRITE 'A' DISK ACCESSED  RC=36
DMSRRV070E INVALID PARAMETER 'parameter'  RC=24
DMSRRV097E NO 'SYSRES' VOLUME ACTIVE  RC=36
DMSRRV098E NO MODULE NAME SPECIFIED  RC=24
DMSRRV099E CMS/DOS ENVIRONMENT NOT ACTIVE  RC=40
DMSRRV105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSRRV113S DISK (cuu) NOT ATTACHED  RC=100
DMSRRV411S INPUT ERROR CODE 'nn' ON '{SYSRES|SYSRLB}'  RC=100
```

RUN

Use the RUN EXEC procedure to initiate a series of functions on a file depending on the filetype. The RUN command can select or combine the procedures required to compile, load, or start execution of the specified file. The format of the RUN command is:

```
| RUN  |  fn [ft [fm]] [(args...[) ]]                              |
```

where:

fn is the filename of the file to be manipulated.

ft is the filetype of the file to be manipulated. If filetype is nct specified, a search is made for a file with the specified filename and the filetype of EXEC, MODULE, or TEXT (the search is performed in that crder). If the filetype of an input file for a language processor is specified, the language processor is invoked to compile the source statements and produce a TEXT file. If no compilation errors are found, LOAD and START may then be called to initiate program execution. The valid filetypes and resulting action fcr this command are:

Filetype _Action_

EXEC The EXEC processor is called to process the file.

MODULE The LOADMOD command is issued to load the program into storage and the START command tegins execution of the program at the entry point equal to fn.

TEXT The LOAD command brings the file intc storage in an executable format and the START command executes the program beginning at the entry point named by fn.

FORTRAN The FORTRAN processor module that is called is FORTRAN, FORTGI, GOFORT, or FORTHX, whichever is found first. Object text successfully compiled by the FORTGI or FORTHX processors will be loaded and executed.

FORTTEST The FORTRAN processor module that is called is either FORTRAN or FORTGI, whichever is found first. The processor is called with the TEST option.

TESTFORT The TESTFORT module is called to initiate FORTRAN Interactive Debug and will process a TEXT file that has been compiled with the TEST option.

FREEFORT The GOFORT module is called to process the file.

COBOL The COBOL processor module that is called is COBOL cr TESTCOB, whichever is found first. After successful compilation, the program text will be loaded and executed.

PLI
PLIOPT The PLIOPT processor module is called to process the file. After successful compilation, the program text will be loaded and executed.

fm is the filemode of the file to be manipulated. If this field is specified, a filetype must be specified. If fm is not specified, the default search order is used to search your disks for the file.

args are arguments you want to pass to your program. You can specify up to 13 arguments in the RUN command, provided they fit on a single input line. Each argument is left-justified, and any argument more than eight characters long is truncated on the right.

Usage Notes

1. The RUN command is an EXEC file; if you want to execute it from within an EXEC, you must use the EXEC command.

2. If you are executing an EXEC file, the arguments you enter on the RUN command line are assigned to the variable symbols &1, &2, and so on.

3. If you are executing a TEXT or MODULE file, or compiling and executing a program, the arguments are placed in a parameter list and passed to your program when it executes. The arguments are placed in a series of doublewords in storage, terminated by X'FF'. If you enter:

   ```
   run myprog (charlie dog
   ```

   the arguments *, CHARLIE, and DOG are placed in doublewords in a parameter list, and the address of the list is in register 1 when your program receives control.

   Note: You cannot use the argument list to override default options for the compilers or for the LOAD or START commands.

4. The RUN command is not designed for use with CMS/DCS.

5. The RUN EXEC cannot be used for COBOL and PL/I programs that require facilities not supported under CMS. For specific language support limitations, see *VM/370 Planning and System Generation Guide*.

Responses

Any responses are from the programs or procedures that executed within the RUN EXEC.

Error Messages and Return Codes

```
DMSRUN001E NO FILENAME SPECIFIED  RC=24
DMSRUN002E FILE['fn [ft [fm]]'] NOT FOUND  RC=28
DMSRUN048E INVALID MODE 'fm'  RC=24
DMSRUN070E INVALID PARAMETER 'parameter'  RC=24
DMSRUN999E NO [ft] PROCESSOR FOUND  RC=28
```

SET

Use the SET command to establish, turn off, or reset a particular function in your CMS virtual machine. Only one function may be specified per SET command. The format of the SET command is:

```
┌─────┬──────────────────────────────────────────────────────────────────┐
│ SET │ function                                                         │
│     │                                                                  │
│     │             ┌                    ┐   ┌           ┐               │
│     │ functions:  │BLIP string[(count)]│   │RDYMSG LMSG│               │
│     │             │BLIP ON             │   │RDYMSG SMSG│               │
│     │             │BLIP OFF            │   └           ┘               │
│     │             └                    ┘                               │
│     │                                                                  │
│     │                       ┌            ┐   ┌      ┌     ┐ ┐          │
│     │ [LDRTBLS nn]          │RELPAGE ON  │   │INPUT │ a xx│ │          │
│     │                       │RELPAGE OFF │   │      │xx yy│ │          │
│     │                       └            ┘   └      └     ┘ ┘          │
│     │                                      [OUTPUT [xx a] ]            │
│     │ ┌           ┐         ┌            ┐   ┌          ┐              │
│     │ │ABBREV ON  │         │REDTYPE ON  │   │IMPEX ON  │              │
│     │ │ABBREV OFF │         │REDTYPE OFF │   │IMPEX OFF │              │
│     │ └           ┘         └            ┘   └          ┘              │
│     │ ┌           ┐         ┌            ┐   ┌             ┐           │
│     │ │IMPCP ON   │         │PROTECT ON  │   │AUTOREAD ON  │           │
│     │ │IMPCP OFF  │         │PROTECT OFF │   │AUTOREAD OFF │           │
│     │ └           ┘         └            ┘   └             ┘           │
│     │                                                                  │
│     │ ┌        ┌CMSDOS  ┐           ┐    ┌         ┌CMSDOS  ┐ ┐        │
│     │ │SYSNAME │CMSVSAM │ entryname │    │NONSHARE │CMSVSAM │ │        │
│     │ │        │CMSAMS  │           │    │         │CMSAMS  │ │        │
│     │ └        └CMSSEG  ┘           ┘    └         └CMSSEG  ┘ ┘        │
│     │                                                                  │
│     │ CMS/DOS functions:                                               │
│     │ ┌                           ┐  ┌           ┐                     │
│     │ │DOS ON [mode [(VSAM[) ]]]│    │DOSLNCNT nn│                     │
│     │ │DOS OFF                    │  └           ┘                     │
│     │ └                           ┘                                    │
│     │                                                                  │
│     │ ┌              ┐ ┌             ┐                                 │
│     │ │UPSI nnnnnnnn │ │DOSPART nnnnK│                                 │
│     │ │UPSI OFF      │ │DOSPART OFF  │                                 │
│     │ └              ┘ └             ┘                                 │
└─────┴──────────────────────────────────────────────────────────────────┘
```

where:

Functions:

BLIP string[(count)]
: defines the characters that are displayed at the terminal to indicate every two seconds of virtual interval timer time. This time is made up of virtual processor time plus, if the REALTIMER option is in effect, self-imposed wait time. Blips may also be caused by the execution of the STIMER macro.

You can define up to eight characters as a blip string; if you want trailing blanks, you must specify count. ON and OFF must not be used as BLIP characters.

BLIP ON
: sets the BLIP character string to its default, which is a string of nonprintable characters. ON is the default for typewriter devices. The default BLIP character provides no visual or audio-visual signal on a 3767 terminal. You must define a BLIP character for a 3767 if you want the BLIP function.

BLIP OFF turns off BLIP. OFF is the default for graphics devices.

Note: The BLIP operand will be ignored when issued from the CMS batch machine.

RDYMSG LMSG
indicates that the standard CMS ready message, including current and elapsed time, is used. The format of the standard Ready message is:

```
R; T=s.mm/s.mm hh:mm:ss
```

where the virtual processor time, real processor time, and clock time are listed.

RDYMSG SMSG
indicates that a shortened form of the CMS ready message (R;) which does not include the time is used.

LDRTBLS nn
defines the number (nn) of pages of storage to be used for loader tables. By default, a virtual machine having up to 384K of addressable real storage has two pages of loader tables; a larger virtual machine has three pages. Each loader table page has a capacity of 204 external names. During LOAD and INCLUDE command processing, each unique external name enccuntered in a TEXT deck is entered in the loader table. The LOAD command clears the table before reading TEXT files; INCLUDE does not. This number can be changed with the SET LDRTBLS nn command provided that: (1) nn is a decimal number between 0 and 128, and (2) the virtual machine has enough storage available to allow nn pages to be used for loader tables. If these two conditions are met, nn pages are set aside for loader tables. If you plan to change the number cf pages allocated for loader tables, you should deallocate storage at the high end of storage so that the storage for the loader tables may be obtained from that area. Usually, you can deallocate storage by releasing one or more of the disks that were accessed.

RELPAGE ON
releases page frames of storage and sets them to binary zercs after the following commands complete execution: ASSEMBLE, COPYFILE, COMPARE, EDIT, MACLIB, SORT, TXTLIB, UPDATE, and the program product language processors supported by VM/370. These processors are listed in the VM/370 Introduction.

RELPAGE OFF
does not release pages of storage after the commands listed in the RELPAGE ON description complete execution. Use the SET RELPAGE OFF function when debugging or analyzing a problem so that the storage used is not released and can be examined.

INPUT a xx
translates the specified character a to the specified hexadecimal code xx for characters entered from the terminal.

INPUT xx yy
allcws you to reset the hexadecimal code xx to the specified hexadecimal code yy in your translate table.

Note: If you issue SET INPUT and SET OUTPUT commands for the same characters, issue the SET OUTPUT command first.

INPUT returns all characters to their default translation.

OUTPUT xx a

translates the specified hexadecimal representation xx to the specified character "a" for all xx characters displayed at the terminal.

OUTPUT returns all characters to their default translation.

Note: Output translation does not occur for SCRIPT files when the SCRIPT command output is directed to the terminal, ncr when you use the CMS editor on a display terminal in display mode.

ABBREV ON

accepts system and user abbreviations for system commands. The SYNONYM command makes the system and user abbreviations available.

ABBREV OFF

accepts only the full system command name or the full user synonym (if one is available) for system commands.

For a discussion of the relationship of the SET ABBREV and SYNONYM commands, refer to the SYNONYM command description.

REDTYPE ON

types CMS error messages in red for certain terminals equipped with the appropriate terminal feature and a two-color ribbon. Supported terminals are described in the VM/370 Terminal User's Guide.

REDTYPE OFF

suppresses red typing of error messages.

IMPEX ON

treats EXEC files as commands; an EXEC file is invoked when the filename of the EXEC file is entered.

IMPEX OFF

does not consider EXEC files as commands. You must issue the EXEC command to execute an EXEC file.

IMPCP ON

passes command names that CMS does not recognize to CP; that is, unknown commands are considered to be CP commands.

IMPCP OFF

generates an error message at the terminal if a command is not reccgnized by CMS.

PROTECT ON

protects the CMS nucleus against writing in its storage area.

PROTECT OFF

does nct protect the storage area containing the CMS nucleus.

AUTOREAD ON

specifies that a console read is to be issued immediately after command execution. ON is the default for nondisplay, ncnbuffered terminals.

AUTOREAD OFF

specifies that you do not want a console read to be issued until you press the Enter key or its equivalent. OFF is the default for display terminals because the display terminal does not lock, even when there is no READ active for it.

Note: If you disconnect from one type of terminal and reconnect on another type, the AUTOREAD status remains unchanged.

```
        (CMSDOS )
SYSNAME )CMSVSAM( entryname
        )CMSAMS (
        (CMSSEG )
```

allows you to replace a saved system name entry in the SYSNAMES table with the name of an alternative, or backup system. A separate SET SYSNAME command must be issued for each name entry to be changed. CMSDOS, CMSVSAM, CMSAMS, and CMSSEG are the default names assigned to the systems when the CMS system is generated.

```
         (CMSDOS )
NONSHARE )CMSVSAM(
         )CMSAMS (
         (CMSSEG )
```

specifies that you want your own nonshared copy of a normally shared named system.

CMS/DOS Functions:

The following functions describe the SET operands that apply to the CMS/DOS environment.

DOS ON places your CMS virtual machine in the CMS/DOS environment. The logical unit SYSLOG is assigned to your terminal.

| mode specifies the mode letter at which the DOS/VSE system residence is accessed; the logical assignment of SYSRES is made for the indicated mode letter.

VSAM specifies that you are going to use the AMSERV command or you are going to execute programs to access VSAM data sets.

DOS OFF returns your virtual machine to the normal CMS environment. All previously assigned system and programmer logical units are unassigned.

DOSLNCNT nn

specifies the number of SYSLST lines per page. nn is an integer from 30 to 99.

UPSI nnnnnnnn

sets the UPSI (User Program Switch Indicator) byte to the specified bit string of 0's and 1's. If you enter fewer than eight digits, the UPSI byte is filled in from the left and zero-padded to the right. If you enter an "x" for any digit, the corresponding bit in the UPSI byte is left unchanged.

UPSI OFF resets the UPSI byte to binary zeros.

DOSPART nnnnnK

specifies the size of the virtual partition in which you want a program to execute. The value, nnnnnK, may not exceed the amount of user free storage available in your virtual machine. You should use this function only when you can control the performance of a particular program by reducing the amount of available virtual storage.

Note: In rare circumstances, it may happen that when a program is executed, the amount of storage available is less than the current DOSPART. Then, only the amount of storage available is obtained; no message is issued.

DOSPART OFF
specifies that you no longer want to control your virtual machine partition size. When the DOSPART setting is OFF, CMS computes the partition size whenever a program is executed.

Usage Notes

1. If you issue the SET command specifying an invalid function and the implied CP function is in effect, you may receive message DMKCFC003E.

2. If an invalid SET command function is specified from an EXEC and the implied CP function is in effect, then the return code is -0003.

Responses

None. To determine or verify the setting of a function, use the QUERY command.

Messages and Return Codes

DMSLIO002I	FILE 'fn' TXTLIB NOT FOUND RC=0
DMSSET014E	INVALID FUNCTION 'function' RC=24
DMSSET026E	INVALID PARAMETER 'parameter' FOR 'function' FUNCTION RC=24
DMSSET031E	LOADER TABLES CANNOT BE MODIFIED RC=40
DMSSET047E	NO FUNCTION SPECIFIED RC=24
DMSSET048E	INVALID MODE 'mode' RC=24
DMSSET050E	PARAMETER MISSING AFTER 'function' RC=24
DMSSET061E	NO TRANSLATION CHARACTER SPECIFIED RC=24
DMSSET070E	INVALID PARAMETER 'parameter' RC=24
DMSSET098W	CMS OS SIMULATION NOT AVAILABLE RC=4
DMSSET099E	CMS/DOS ENVIRONMENT NOT ACTIVE RC=40
DMSSET100W	SYSTEM NAME 'name' NOT AVAILABLE RC=4
DMSSET142S	SAVED SYSTEM NAME 'name' INVALID RC=24
DMSSET333E	nnnnnK PARTITION TOO LARGE FOR THIS VIRTUAL MACHINE RC=24
DMSSET400S	SYSTEM 'sysname' DOES NOT EXIST RC=44
DMSSET401S	V.M. SIZE (size) CANNOT EXCEED 'DMSDOS' START ADDRESS (address) RC=104
DMSSET410S	CONTROL PROGRAM ERROR INDICATION 'retcode' RC=nnn *Note*: In RC=nnn, the nnn represents the actual error code generated by CP.
DMSSET444E	VOLUME 'label' IS NOT A DOS SYSRES RC=32

SORT

Use the SORT command to read fixed-length records from a CMS input file, arrange them in ascending EBCDIC order according to specified sort fields, and create a new file containing the sorted records. The format of the SORT command is:

```
| SORT |   fileid1  fileid2                                      |
```

where:

fileid1 is the file identifier (filename, filetype, filemode) of the file containing the records to be sorted.

fileid2 is the file identifier (filename, filetype, filemode) of the new output file to contain the sorted records.

Usage Note

The input and output files must not have the same file identifiers, since SORT cannot write the sorted output back into the space occupied by the input file. If a file with the same name as the output file already exists, the old file is erased.

Entering Sort Control Fields: After the SORT command is entered, CMS responds with the following message on the terminal:

```
DMSSRT604R ENTER SORT FIELDS:
```

You should respond by entering one or more pairs of numbers of the form "xx yy" separated by one or more blanks. Each "xx" is the starting character position of a sort field within each input record and "yy" is the ending character position. The leftmost pair of numbers denotes the major sort field. The number of sort fields is limited to the number of fields you can enter on one line. The records can be sorted on up to a total of 253 positions.

Virtual Storage Requirements for Sorting: The sorting operation takes place with two passes of the input file. The first pass creates an ordered pointer table in virtual storage. The second pass uses the pointer table to read the input file in a random manner and write the output file. Therefore, the size of storage and the size and number of sort fields are the limiting factors in determining the number of records that can be sorted at any one time. An estimate of the maximum number of records that can be sorted is:

```
             VMSIZE - 132K
      NR =   -------------
              14 + NC
```

where: NR is the estimated maximum number of input records; NC is the total number of characters in the defined sort fields; VMSIZE is the storage size of the virtual machine; and 132K is the size of the resident CMS nucleus. For example, enter the command and respond to the prompting message:

```
     sort name address a1 sortedna address b1

     DMSSRT604R ENTER SORT FIELDS:

     1 10 25 28
```

The records in the NAME ADDRESS file are sorted on positions 1-10 and 25-28. The sorted output is written into the newly created file SORTEDNA ADDRESS. If you have a 320K virtual machine, you can sort a maximum of 6875 records.

```
          VMSIZE-132K   320K-132K   188K   192,512
     NR = ----------- = --------- = ---- = ------- = 6875
           14 + NC       14 + 14     28      28
```

Responses

DMSSRT604R ENTER SORT FIELDS:

> You are requested to enter SORT control fields. You should enter them in the form described previously in "Entering Sort Contrcl Fields."

Other Messages and Return Codes

```
DMSSRT002E FILE 'fm ft fm' NOT FOUND  RC=28
DMSSRT009E COLUMN 'col' EXCEEDS RECORD LENGTH  RC=24
DMSSRT019E IDENTICAL FILEIDS  RC=24
DMSSRT034E FILE 'fn ft fm' IS NOT FIXED LENGTH  RC=32
DMSSRT037E DISK 'mode' IS READ/ONLY  RC=36
DMSSRT053E INVALID SORT FIELD PAIR DEFINED  RC=24
DMSSRT054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSSRT062E INVALID * IN FILEID  RC=20
DMSSRT063E NO LIST ENTERED  RC=40
DMSSRT070E INVALID PARAMETER 'parameter'  RC=24
DMSSRT104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSSRT105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSSRT212E MAXIMUM NUMBER OF RECORDS EXCEEDED  RC=40
```

SSERV

Use the SSERV command in CMS/DOS to copy, display, print, or punch a book from a DOS/VSE source statement library. The format of the SSERV command is:

```
┌──────────────────────────────────────────────────────────────────────┐
│          │                    ┌    ┐                                  │
│ SSERV    │ sublib bookname    │ ft │  [(options... [)]]               │
│          │                    │COPY│                                  │
│          │                    └    ┘                                  │
│          │                            options:                        │
│          │                                 [DISK]   [PRINT]           │
│          │                                 [PUNCH   [TERM]            │
└──────────────────────────────────────────────────────────────────────┘
```

where:

sublib specifies the source statement sublibrary in which the book is cataloged.

bookname specifies the name of the book in the DOS private or system source statement sublibrary. The private library, if any, is searched before the system library.

ft specifies the filetype of the file to be created on your A-disk. ft defaults to COPY if a filetype is not specified. The filename is always the same as the bookname.

Options: You may enter as many options as you wish, depending upon the functions you want to perform.

DISK copies the book to a CMS file.

PUNCH punches the book on the virtual punch.

PRINT spools a copy of the book to your virtual printer.

TERM displays the book on your terminal.

Usage Notes

1. If you want to copy books from private libraries, you must issue an ASSGN command for the logical unit SYSSLB and identify the library on a DLBL command line using a ddname of IJSYSSL.

   If you want to copy books from the system library, you must have entered the CMS/DOS environment specifying the mode letter of the system residence volume.

2. You should not use the SSERV command to copy books from macro (E) sublibraries, since they are in "edited" (that is, compressed) form. Use the ESERV command to copy and de-edit macros from a macro (E) sublibrary.

Responses

When you use the TERM option, the specified book is displayed at the terminal.

Messages and Return Codes

```
DMSSRV003E INVALID OPTION 'option'  RC=24
DMSSRV004E BOOK 'subl.book' NOT FOUND  RC=28
DMSSRV006E NO READ/WRITE 'A' DISK ACCESSED  RC=36
DMSSRV070E INVALID PARAMETER 'parameter'  RC=24
DMSSRV097E NO 'SYSRES' VOLUME ACTIVE  RC=36
DMSSRV098E NO BOOK NAME SPECIFIED  RC=24
DMSSRV099E CMS/DOS ENVIRONMENT NOT ACTIVE  RC=40
DMSSRV105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSSRV113S DISK (cuu) NOT ATTACHED  RC=100
DMSSRV411S INPUT ERROR CODE 'nn' ON '{SYSRES|SYSSLB}'  RC=100
DMSSRV194S BOOK 'subl.book' CONTAINS BAD RECORDS  RC=100
```

START

Use the START command to begin execution of CMS, OS, or DOS programs that were previously loaded or fetched. The format of the START command is:

```
┌──────────────────────────────────────────────────────────────────┐
│         │    ┌                    ┐                              │
│  START  │    │ entry  [args...]   │  [(option[)]]                │
│         │    │ *                  │    option:                   │
│         │    └                    ┘      NO                      │
└──────────────────────────────────────────────────────────────────┘
```

where:

entry passes control to the control section name or entry point name at execution time. The operand, entry, may be a filename only if the filename is identical to a control section name or an entry point name.

* passes control to the default entry point. See the discussion of the LOAD command for a discussion of the default entry point selection.

args... are arguments to be passed to the started program. If user arguments are specified, the entry or * operands must be specified; otherwise, the first argument is taken as the entry point. Arguments are passed to the program via general register 1. The entry operand and any arguments become a string of doublewords, one argument per doubleword, and the address of the list is placed in general register 1.

Option:

NO suppresses execution of the program. Linkage editor and loader functions are performed and the program is in storage ready to execute, but control is not given to the program.

Usage Notes:

1. Any undefined names or references specified in the files loaded into storage are defined as zero. Thus, if there is a call or branch to a subroutine from a main program, and if the subroutine has never been loaded, the call or branch transfers control to location zero of the virtual machine at execution time.

2. Do not use the START command for programs that are generated via the GENMOD command with the NOMAP option. The START command does not execute properly for such programs.

Responses

DMSLIO740I EXECUTION BEGINS...

is displayed when the designated entry point is validated.

This message is suppressed if CMS/DOS is active and the COMP option is specified in the FETCH command.

Error Messages and Return Codes

DMSLIO021E ENTRY POINT 'name' NOT FOUND RC=40
DMSLIO055E NO ENTRY POINT DEFINED RC=40

STATE/STATEW

Use the STATE command to verify the existence of a CMS, OS, or DOS file on any accessed disk; use the STATEW command to verify the existence cf a CMS, OS, or DOS file on any accessed read/write disk. The formats cf the STATE and STATEW commands are:

```
┌──────────────────────────────────────────────────────────────────────┐
| {STATE }  |  {fn} {ft}{[fm]}                                         |
| {STATEW}  |  { *} { *}{  * }                                         |
└──────────────────────────────────────────────────────────────────────┘
```

where:

fn   is the filename of the file whose existence is to be verified. If fn is specified as *, the first file found satisfying the rest cf the fileid is used.

ft   is the filetype of the file whose existence is to be verified. If ft is specified as *, the first file found satisfying the rest cf the fileid is used.

fm   is the filemode of the file whose existence is to be verified. If fm is omitted, or specified as *, all your disks are searched.

Usage Notes:

1. If you issue the STATEW command specifying a file that exists on a read-only disk, you receive error message DMSSTT002E.

2. When you code an asterisk in the fn or ft fields, the search for the file is ended as soon as any file satisfies any of the other conditions. For example, the command:

   ```
   state * file
   ```

   executes successfully if any file on any accessed disk (including the system disk) has a filetype of FILE.

3. To verify the existence of an OS or DOS file when DOS is set OFF, you must issue the FILEDEF command to establish a CMS file identifier for the file. For example, to verify the existence cf the OS file TEST.DATA on an OS C-disk you could enter:

   ```
   filedef check disk check list c dsn test data
   state check list
   ```

   where CHECK LIST is the CMS filename and filetype associated with the OS data set name.

4. To verify the existence of an OS or DOS file when the CMS/DCS environment is active, you must issue the DLEL command to establish a CMS file identifier for the file. For example, to verify the existence of the DOS file TEST.DATA on a DOS C-disk, you could enter:

   ```
   dlbl check c dsn test data
   state file check
   ```

   where FILE CHECK is the default CMS filename and filetype (FILE ddname) associated with the DOS file-id.

5. You can invoke the STATE/STATEW command from the terminal, from an EXEC file, or as a function from a program. If STATE/STATEW is invoked as a function or from an EXEC file that has the &CONTRCL NOMSG option in effect, the message DMSSTTC02E FILE 'fn ft fm' NCT FOUND is not issued.

Responses

The CMS ready message indicates that the specified file exists.

DMSSTT227I PROCESSING VOLUME 'no' IN DATA SET 'data set name'

> The specified data set has multiple volumes; the volume being processed is shown in the message. The STATE command treats end-of-volume as end-of-file and there is no end-of-volume switching.

DMSSTT228I USER LABELS BYPASSED ON DATA SET 'data set name'

> The specified data set has disk user labels; these labels are skipped.

Error Messages and Return Codes

```
DMSSTT002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSSTT048E INVALID MODE 'mode'  RC=24
DMSSTT054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSSTT062E INVALID 'char' IN FILEID 'fn ft'  RC=20
DMSSTT069E DISK 'mode' NOT ACCESSED  RC=36
DMSSTT070E INVALID PARAMETER 'parameter'  RC=24
DMSSTT229E UNSUPPORTED OS DATA SET, ERROR 'code'  RC=code
```

SVCTRACE

Use the SVCTRACE command to trace and record information about supervisor calls occurring in your virtual machine. The format of the SVCTRACE command is:

```
| SVCTrace  |   {ON }
|           |   {OFF}
```

where:

ON starts tracing all SVC instructions issued within CMS.

OFF stops SVC tracing.

Usage Notes

1. The trace information recorded on the printer includes:

   - The virtual storage location of the calling SVC instruction and the name of the called program or routine

   - The normal and error return addresses

   - The contents of the general registers both before the SVC-called program is given control and after a return from that program

   - The contents of the general registers when the SVC handling routine is finished processing

   - The contents of the floating-point registers before the SVC-called program is given control and after a return from that program

   - The contents of the floating-point registers when the SVC handling routine is finished processing

   - The parameter list passed to the SVC

2. To terminate tracing previously established by the SVCTRACE command, issue the HO or SVCTRACE OFF commands. SVCTRACE OFF and HO cause all trace information recorded, up to the point they are issued, to be printed on the virtual spooled printer. Cn typewriter terminals SVCTRACE OFF can be issued only when the keyboard is unlocked to accept input to the CMS command environment. To terminate tracing at any other point in system processing, HO must be issued. To suspend tracing temporarily during a session, interrupt processing and enter the Immediate command SO (Suspend Tracing). To resume tracing that was suspended with the SO command, enter the Immediate command RO (Resume Tracing).

   If you issue the CMS Immediate command HX or you log off the VM/370 system before termination of tracing previously set by the SVCTRACE command, the switches are cleared automatically and all recorded trace information is printed on the virtual spooled printer.

   If a user timer exit is activated while SVCTRACE is active, SVCTRACE is disabled for the duration of the timer exit. Any SVCs issued during the timer exit are not reflected in the SVCTRACE listing.

3. When tracing on a virtual machine with only one printer, the trace data is intermixed with other data sent to the virtual printer.

Responses

A variety of information is printed whenever the:

```
    SVCTRACE ON
```

command is issued.

The first line of trace output starts with a dash or plus sign or an asterisk (- or + or *). The format of the first line of trace output is:

```
⎧ - ⎫ N/D = xxx/dd name FROM loc OLDPSW = psw1 GOPSW = psw2 [RC=rc]
⎨ + ⎬
⎩ * ⎭
```

where:

- indicates information recorded before processing the SVC.

+ indicates information recorded after processing the SVC, unless the asterisk (*) applies.

* indicates information recorded after processing a CMS SVC that had an error return.

N/D is an abbreviation for SVC number and depth (or level).

xxx is the number of the SVC call (they are numbered sequentially).

dd is the nesting level of the SVC call.

name is the macro or routine being called.

loc is the program location from which the SVC was issued.

psw1 is the PSW at the time the SVC was called.

psw2 is the PSW with which the routine being called is invoked, if the first character of this line is a dash (-). If the first character of this line is a plus sign or asterisk (+ or *), PSW2 represents the PSW that returns control to the user.

rc is the return code from the SVC handling routine in general register 15. This field is omitted if the first character of this line is a dash (-), or if this is an OS SVC call. For a CMS SVC, this field is 0 if the line begins with a plus sign (+), and nonzero for an asterisk (*). Also, this field equals the contents of R15 in the "GPRS AFTER" line.

The next two lines of output are the contents of the general registers when control is passed to the SVC handling routine. This output is identified at the left by ".GPRSB". The format of the output is:

```
    .GPRSB = h h h h h h h h *dddddddd*
           = h h h h h h h h *dddddddd*
```

where h represents the contents of a general register in hexadecimal format and d represents the EBCDIC translation of the contents of a general register. The contents of general registers 0 through 7 are printed on the first line, with the contents of registers 8 through F on the second line. The hexadecimal contents of the registers are printed first, followed by the EBCDIC translation. The EBCDIC translation is preceded and followed by an asterisk(*).

The next line of output is the contents of general registers 0, 1, and 15 when control is returned to your program. The output is identified at the left by ".GPRS AFTER :". The format of the output is:

```
.GPRS AFTER : R0-R1 = h h *dd* R15 = h *d*
```

where h represents the hexadecimal contents of a general register and d is the EBCDIC translation of the contents of a general register. The only general registers that CMS routines alter are registers 0, 1, and 15 so only those registers are printed when control returns to your program. The EBCDIC translation is preceded and followed by an asterisk (*).

The next two lines of output are the contents of the general registers when the SVC handling routine is finished processing. This output is identified at the left by ".GPRSS." The format of the output is:

```
.GPRSS = h h h h h h h h *dddddddd*
       = h h h h h h h h *dddddddd*
```

where h represents the hexadecimal contents of a general register and d represents the EBCDIC translation of the contents of a general register. General registers 0 through 7 are printed on the first line with registers 8 through F on the second line. The EBCDIC translation is preceded and followed by an asterisk (*).

The next line of output is the contents of the calling routine's floating-point registers. The output is identified at the left by ".FPRS". The format of the output is:

```
.FPRS = f  f  f  f  *gggg*
```

where f represents the hexadecimal contents of a floating-point register and g is the EBCDIC translation of a floating-point register. Each floating point register is a doubleword; each f and g represents a doubleword of data. The EBCDIC translation is preceded and followed by an asterisk (*).

The next line of output is the contents of floating-point registers when the SVC handling routine is finished processing. The output is identified by ".FPRSS" at the left. The format of the output is:

```
.FPRSS = f  f  f  f  *gggg*
```

where f represents the hexadecimal contents of a floating-point register and g is the EBCDIC translation. Each floating-point register is a doubleword and each f and g represents a doubleword of data. The EBCDIC translation is preceded and followed by an asterisk (*).

The last two lines of output are printed only if the address in register 1 is a valid address for the virtual machine. If printed, the output is the parameter list passed to the SVC. The output is identified by ".PARM" at the left. The output format is:

```
.PARM = h h h h h h h h *dddddddd*
      = h h h h h h h h *dddddddd*
```

where h represents a word of hexadecimal data and d is the EBCDIC translation. The parameter list is found at the address contained in register 1 before control is passed to the SVC handling program. The EBCDIC translation is preceded and followed by an asterisk (*).

Figure 19 summarizes the types of SVC trace output.

Identification	Comments
$\left\{\begin{array}{c}-\\+\\*\end{array}\right\}$ N/D	The SVC and the routine that issued the SVC.
.GPRSB	Contents of general registers when control is passed to the SVC handling routine.
.GPRS AFTER	Contents of general registers 0, 1, and 15 when control is returned to your program.
.GPRSS	Contents of the general registers when the SVC handling routine is finished processing.
.FPRS	Contents of floating-point registers before the SVC-called program is given control and after returning from that program.
.FPRSS	Contents of the floating-point registers when the SVC handling routine is finished processing.
.PARM	The parameter list, when one is passed to the SVC.

Figure 19. Summary of SVC Trace Output Lines

Messages and Return Codes

```
DMSOVR014E INVALID FUNCTION 'function'  RC=24
DMSOVR047E NO FUNCTION SPECIFIED  RC=24
DMSOVR104S ERROR 'nn' READING FILE 'DMSOVR MODULE' ON DISK  RC=100
DMSOVR109S VIRTUAL STORAGE CAPACITY EXCEEDED  RC=104
```

SYNONYM

Use the SYNONYM command to invoke a table of synonyms to be used with, or in place of, CMS and user-written command names. You create the table yourself using the CMS editor. The form for specifying the entries for the table is described under "The User Synonym Table."

The names you define can be used either instead of or in conjunction with the standard CMS command truncations. However, no matter what truncations, synonyms, or truncations of the synonyms are in effect, the full real name of the command is always accepted. The format of the SYNONYM command is:

```
┌───────────────────────────────────────────────────────────────────┐
│             │     ┌   ┌         ┌  ┐┐┐                            │
│  SYNonym    │     │fn │SYNONYM  │fm│││ [(options...[)]]           │
│             │     │   │         │A1│││                            │
│             │     └   └         │* │┘┘                            │
│             │                   └  ┘                              │
│             │                                                     │
│             │                           ┌     ┐                   │
│             │                 options:  │STD  │  [CLEAR]          │
│             │                           │NOSTD│                   │
│             │                           └     ┘                   │
└───────────────────────────────────────────────────────────────────┘
```

where:

fn is the filename of the file containing your synonyms table.

fm is the filemode of the file containing your synonyms; if omitted, your A-disk and its extensions are searched. If you specify fm, you must enter the keyword, SYNONYM. If you specify fm as an asterisk (*), all disks are searched for the specified SYNONYM file.

Options:

STD specifies that standard CMS abbreviations are accepted.

NOSTD standard CMS abbreviations are not to be accepted. (The full CMS command and the synonyms you defined can still be used.)

CLEAR removes any synonym table set by a previously entered SYNONYM command.

Usage Notes

1. If you enter the SYNONYM command with no operands, the system synonym table and the user synonym table (if one exists) are listed.

2. The SET ABBREV ON or OFF command, in conjunction with the SYNONYM command, determines which standard and user-defined forms of a particular CMS command are acceptable.

THE USER SYNONYM TABLE

You create the synonym table using the CMS editor. The table must be a file with the filetype SYNONYM. The file consists of 80-byte fixed-length records in free-form format with columns 73-80 ignored. The format for each record is:

```
systemcommand usersynonym count
```

where:

systemcommand
    is the name of the CMS command or MODULE or EXEC file for which you are creating a synonym.

usersynonym
    is the synonym you are assigning to the command name. When you create the synonym, you must follow the same syntax rules as for commands; that is, you must use the character set used to create commands, the synonym may be no longer than eight characters, and so on.

count is the minimum number of characters that must be entered for the synonym to be accepted by CMS. If omitted, the entire synonym must be entered (see the following example).

A table of command synonyms is built from the contents of this file. You may have several synonym files but only one may be active at a time. For example, if the synonym file named MYSYN contains:

```
MOVEFILE  MVIT
```

then, after you have issued the command:

```
synonym mysyn
```

the synonym MVIT can be entered as a command name to execute the MOVEFILE command. It cannot be truncated since no count is specified. If MYSYN SYNONYM contains:

```
ACCESS GETDISK 3
```

then, the syncnyms GET, GETD, GETDI, GETDIS, or GETDISK can be entered as the command name instead of ACCESS.

If you have an EXEC file named TDISK, you might have a synonym entry:

```
TDISK TDISK 2
```

so that you can invoke the EXEC procedure by specifying the truncation TD.

The Relationship between the SET ABBREV and SYNONYM Commands

The default values of the SET and SYNONYM commands are such that the system syncnym abbreviation table is available unless otherwise specified.

The system synonym abbreviation table for the FILEDEF command states that FI is the minimum truncation. Therefore, the acceptable abbreviations for FILEDEF are: FI, FIL, FILE, FILED, FILEDE, and FILEDEF. The system synomym abbreviation table is available whenever both SET ABBREV ON and SYNONYM (STD) are in effect.

If you have a synonym table with the file identification USERTAB SYNONYM A, that has the entry:

```
FILEDEF USENAME 3
```

then, USENAME is a synonym for FILEDEF, and acceptable truncations of USENAME are: USE, USEN, USENA, USENAM, and USENAME. The user synonym abbreviation table is available whenever both SET ABBREV ON and SYNONYM USERTAB are specified.

No matter what synonyms and truncations are defined, the full real name of the command is always in effect.

Figure 20 lists the forms of the system command and user synonyms available for the various combinations of the SET ABBREV and SYNONYM commands.

Responses

When you enter the SYNONYM command with no operands, the synonym table(s) currently in effect are displayed.

```
SYSTEM      USER        SHORTEST
COMMAND     SYNONYM     FORM (IF ANY)
--------    -------     ------------
   .           .           .
   .           .           .
   .           .           .
```

This response is the same as the response to the command QUERY SYNONYM ALL.

DMSSYN711I NO SYSTEM SYNONYMS IN EFFECT

This response is displayed when you issue the SYNONYM command with no operands after the command SYNONYM (NOSTD) has been issued.

DMSSYN712I NO SYNONYMS (DMSINA NOT IN NUCLEUS)

The system routine which handles SYNONYM command processing is not in the system.

Other Messages and Return Codes

```
DMSSYN002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSSYN003E INVALID OPTION 'option'  RC=24
DMSSYN007E FILE 'fn ft fm' NOT FIXED, 80 CHAR RECORDS  RC=32
DMSSYN032E INVALID FILETYPE 'ft'  RC=24
DMSSYN056E FILE 'fn ft fm' CONTAINS INVALID RECORD FORMATS  RC=32
DMSSYN066E 'option AND 'option' ARE CONFLICTING OPTIONS  RC=24
DMSSYN104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
```

Options	Acceptable Command Forms	Comments
SET ABBREV ON SYN USERTAB (STD	FI FIL . . . FILEDEF USE USEN . . . USENAME	The ABBREV ON option of the SET command and the STD option of the SYNONYM command make the system table available. The user synonym, USENAME, is available because the synonym table (USERTAB) is specified on the SYNONYM command. The truncations for USENAME are available because SET ABBREV ON was specified with the USERTAB also available.
SET ABBREV OFF SYN USERTAB (STD	FILEDEF USENAME	The user-defined synonym, USENAME, is permitted because the user synonym table (USERTAB) is specified on the SYNONYM command. No system or user truncations are permitted.
SET ABBREV ON SYN USERTAB (NOSTD	FILEDEF USE USEN . . . USENAME	The system synonym table is unavailable because the NOSTD option is specified on the SYNONYM command. The user synonym, USENAME, is available because the user synonym table (USERTAB) is specified on the SYNONYM command and the truncations of USENAME are permitted because SET ABBREV ON is specified with USERTAB also available.
SET ABBREV OFF SYN USERTAB (NOSTD	FILEDEF USENAME	The system synonym table is made unavailable either by the SET ABBREV OFF command or by the SYN (NOSTD command. The synonym, USENAME, is permitted because the user-defined synonym table (USERTAB) is specified on the SYNONYM command. The truncations for USENAME are not permitted because the SET ABBREV OFF option is in effect.
SET ABBREV ON SYN (CLEAR STD	FI FIL . . . FILEDEF	The user-defined table is now unavailable. The system synonym table is available because both the ABBREV ON option of the SET command and the STD option of the SYNONYM command are specified.
SET ABBREV OFF SYN (CLEAR STD	FILEDEF	Because CLEAR is specified on the SYNONYM command, the synonym and its truncations are no longer available. Either the SET ABBREV OFF command or the SYNONYM (NOSTD command make the system synonym table unavailable.
SET ABBREV ON SYN (CLEAR NOSTD		
SET ABBREV OFF SYN (CLEAR NOSTD		

Figure 20. System and User-Defined Truncations

TAPE

Use the TAPE command to dump CMS-formatted files from disk to tape, load previously dumped files from tape to disk, and perform various control operations on a specified tape drive. Files processed by the TAPE command must be in a unique CMS format. The TAPE command does not process multivolume files. Disk files to be dumped can contain either fixed- or variable-length records. The format of the TAPE command is:

```
┌──────┬──────────────────────────────────────────────────────────────────────┐
│ TAPE │ ┌ DUMP    {fn} {ft} [fm]  [(optionA optionB optionD[)]]             ┐ │
│      │ │         {* } {* } [* ]                                            │ │
│      │ │                                                                   │ │
│      │ │ LOAD    [{fn} {ft} [fm]]  [(optionB optionC optionD[)]]           │ │
│      │ │         [{* } {* } [A ]]                                          │ │
│      │ │                                                                   │ │
│      │ │ SCAN    [ {fn} {ft} ]     [(optionB optionC optionD[)]]           │ │
│      │ │         [ {* } {* } ]                                             │ │
│      │ │                                                                   │ │
│      │ { SKIP    [{fn} {ft}]       [(optionB optionC optionD[)]]           } │
│      │ │         [{* } {* }]                                               │ │
│      │ │                                                                   │ │
│      │ │ DVOL1                     [(optionD optionE[)]]                   │ │
│      │ │ WVOL1 volid [owner]       [(optionD optionE[)]]                   │ │
│      │ │                                                                   │ │
│      │ │ MODESET          [(optionD[)]]                                    │ │
│      │ │                                                                   │ │
│      │ │ tapcmd   [n]     [(optionD[)]]                                    │ │
│      │ └          [1]                                                      ┘ │
│      │                                                                        │
│      │ optionA:   [WTM  ]  [BLKSIZE{4096}]                                    │
│      │            [NOWTM]  [        { 800}]                                   │
│      │                                                                        │
│      │ optionB:   [NOPRint]                                                   │
│      │            [PRint  ]                                                   │
│      │            [Term   ]                                                   │
│      │            [DISK   ]                                                   │
│      │                                                                        │
│      │ optionC:   [EOT   ]                                                    │
│      │            [EOF n ]                                                    │
│      │            [EOF 1 ]                                                    │
│      │                                                                        │
│      │ optionD:   [[TAPn ]]  [7TRACK]  [DEN den]  [TRTCH a]                   │
│      │            [[TAP1 ]]  [9TRACK]                                         │
│      │            [       ]                                                   │
│      │            [[cuu  ]]                                                   │
│      │            [[181  ]]                                                   │
│      │                                                                        │
│      │ optionE:   [REWIND]                                                    │
│      │            [LEAVE ]                                                    │
└──────┴──────────────────────────────────────────────────────────────────────┘
```

where:

```
DUMP ┌fn┐┌ft┐[fm]
     └* ┘└* ┘[* ]
```

dumps one or more disk files to tape. If fn and/or ft is specified as an asterisk (*) all files that satisfy the other file identifier are dumped.

If fm is coded as a letter, that disk and its extensions are searched for the specified file(s). If fm is coded as a letter and number, only files with that mode number and letter (and the extensions of the disk referenced by that fm letter) are dumped. If fm is coded as asterisk (*), all accessed disks are searched for the specified file(s). If fm is not specified, only the A-disk and its extensions are searched.

```
LOAD [┌fn┐┌ft┐[fm]]
     [└* ┘└* ┘[A ]]
```

reads tape files onto disk. If a file identifier is specified, only that one file is loaded. If the option EOF n is specified and no file identifier is entered, n tape files are written to disk. If an asterisk (*) is specified for fn or ft, all files within EOF n that satisfy the other file identifier are loaded.

The files are written to the disk indicated by the filemode letter. The filemode number, if entered, indicates that only files with that filemode number are to be loaded.

```
SCAN [┌fn┐┌ft┐]
     [└* ┘└* ┘]
```

positions the tape at a specified point, and lists the identifiers of the files it scans. Scanning occurs over n tape marks, as specified by the option EOF n (the default is 1 tape file). However, if a file identifier (fn and ft) is specified, scanning stops upon encountering that file; the tape remains positioned ahead of the file.

```
SKIP [┌fn┐┌ft┐]
     [└* ┘└* ┘]
```

positions the tape at a specified point and lists the identifiers of the files it skips. Skipping occurs over n tape marks, as specified by the option EOF n (the default is 1 tape mark). However, if a file identifier (fn and ft) is specified, skipping stops after encountering that file; the tape remains positioned immediately following the file.

MODESET sets the values specified by the DEN, TRACK, and TRTCH options. After initial specification in a TAPE command, these values remain in effect for the virtual tape device until they are changed in a subsequent TAPE command.

```
       ┌ ┐
tapcmd │n│
       │1│
       └ ┘
```

specifies a tape control function (tapcmd) to be executed n times (default is 1 if n is not specified). These functions also work on tapes in a non-CMS format.

Tapcmd	Action
BSF	Backspace n tape marks
BSR	Backspace n tape records
ERG	Erase gap
FSF	Forward-space n tape marks
FSR	Forward-space n tape records
REW	Rewind tape to load point

Tapcmd	Action
RUN	Rewind tape and unload
WTM	Write n tape marks

DVOL1 displays an 80-character VOL1 label in EBCDIC on the user's terminal if such a label exists on the tape. If the first record on the tape is not a VOL1 label, an error message is sent to the user.

WVOL1 volid [owner]
writes a VOL1 label on a tape. All fields are set to the same values they are set to when a VOL1 label is written by the IBM-supplied IEHINITT utility program (see the publication OS/VS2 MVS Utilities for details). The volid is set to the 1- to 6-character volid specified on the command. If the user specifies owner field, it is written in the owner name and address code field of the label. It can be up to eight characters long and left-justified in the 10-byte field in the label. If not specified, the owner field is set to blanks. The WVOL1 option also writes a dummy HDR1 label and tape mark after the VOL1 label.

Options:
If conflicting options are specified, the last one entered is in effect.

WTM writes a tape mark on the tape after each file is dumped.

NOWTM writes a tape mark after each file is dumped, then backspaces over the tape mark so that subsequent files written on the tape are not separated by tape marks.

BLKSIZE 4096
BLKSIZE 800
specifies the size of the tape data block at which the files are to be dumped (not including a five-byte prefix).

NOPRINT does not spool the list of files dumped, loaded, scanned, or skipped to the printer.

PRINT spools the list of files dumped, loaded, scanned, or skipped to the printer.

TERM displays a list of files dumped, loaded, scanned, or skipped at the terminal.

DISK creates a disk file containing the list of files dumped, loaded, scanned, or skipped. The disk file has the file identification of TAPE MAP A5.

EOT reads the tape until an end-of-tape indication is received.

EOF n
EOF 1 reads the tape through a maximum of n tape marks. The default is EOF 1.

TAPn
18n specifies the symbolic tape identification (TAPn) or the actual device address of the tape to be read from or written to where n is 1, 2, 3, or 4. The default is TAP1 or 181. The unit specified by cuu must previcusly have been attached to your CMS virtual machine before any tape I/O operation can be attempted. Only symbolic names TAP1 through TAP4 and virtual device addresses 181 through 184 are supported.

7TRACK specifies a 7-track tape. Odd parity, data convert on, and translate off are assumed unless TRTCH is specified.

9TRACK specifies a 9-track tape.

DEN den is the tape density where den is 200, 556, 800, 1600, or 6250. If 200 or 556 is specified, 7TRACK is assumed. If 1600 or 6250 is specified, 9TRACK is assumed; if 800 is specified, 9TRACK is assumed unless 7TRACK is specified. In the case of either 800/1600 or 1600/6250 dual-density drives, 1600 is the default if the 9TRACK option is specified. If neither the 9TRACK option nor the DEN option is specified, the drive operates at whatever bpi the tape drive was last set.

TRTCH a is the tape recording technique for 7-track tape. If TRTCH is specified, 7TRACK is assumed. One of the following must be specified as "a":

a	Meaning
O	Odd parity, data convert off, translate off
OC	Odd parity, data convert on, translate off
OT	Odd parity, data convert off, translate on
E	Even parity, data convert off, translate off
ET	Even parity, data convert off, translate on

REWIND
LEAVE

are only valid for the DVOL1 and WVOL1 functions. They specify the positioning of a tape after the VOL1 is processed. If REWIND is specified, the tape is rewound and positioned at load point. If LEAVE (the default) is specified, the tape is positioned at the record immediately after the VOL1 label.

Usage Notes

1. Tape records written by the CMS TAPE DUMP command are either 805 bytes long, if the option BLKSIZE is specified as 800; or 4101 bytes long if the BLKSIZE is specified as 4096, or defaulted to 4096. The first character is a binary 2 (X'02'), followed by the characters CMS and a file format byte. For a variable format file, the file format byte is V. For a fixed format file without null blocks, the file format byte is F; otherwise the file format byte is S. In the final record, the character N replaces the file format byte, and the data area contains CMS file directory information. A tape created at 4096-byte block size is not reloadable on a CMS system that does not have the multivalue BLKSIZE option on the TAPE command; however, the 800-byte BLKSIZE option provides backward compatibility to such a system.

2. If a tape file contains a large number of CMS files that would not fit on disk, the tape load operation may terminate if there is not enough disk space to hold the files. To prevent this, when you dump the files, separate logical files by tape marks, then forward space to the appropriate file.

3. Because the CMS file directory is the last record of the file, the TAPE command creates a separate workfile so that backspacing and rereading can be avoided when the disk file is built. If the load criteria is not satisfied, the workfile is erased; if it is satisfied, the workfile is renamed. This workfile is named TAPE CMSUT1, which may exist if a previous TAPE command has abnormally terminated. If the work file is accidentally dumped to tape and subsequently loaded, it appears on your disk as TAPE CMSUT2.

4. The RUN option (rewind and unload) indicates completion before the physical operation is completed. Thus, a subsequent operation to the same physical device may encounter a device busy situation.

5. DVOL1 and WVOL1 are the only TAPE command functions that automatically process tape labels. TAPE DUMP does not automatically write labels on a tape when it writes the dump file, and TAPE LOAD does not recognize tape labels when loading a file.

6. Do not use TAPE DVOL1 for a tape that you suspect to be blank. If you do, and the tape is blank, it will run off the reel.

7. The options for the 8809 tape drive must be 9TRACK and DEN 1600. Note that these are the default values, so you do not need to specify them.

8. For more information on tape file handling, see the *VM/370 CMS User's Guide*.

Responses

DMSTPE701I NULL FILE

A final record was encountered and no prior records were read in a TAPE LOAD operation. No file is created on disk.

If the TERM option is in effect, the following is displayed at the terminal depending on the operation specified:

```
LOADING.....
   fn ft fm
    .  .  .
    .  .  .
    .  .  .

SKIPPING.....
   fn ft fm
    .  .  .
    .  .  .
    .  .  .

DUMPING.....
   fn ft fm
    .  .  .
    .  .  .
    .  .  .
SCANNING.....
   fn ft fm
    .  .  .
    .  .  .
    .  .  .
```

When a tape mark is encountered, the following is displayed at the terminal if the TERM option is specified:

```
END-OF-FILE OR END-OF-TAPE
```

Other Messages and Return Codes

```
DMSTPE002E FILE 'fn ft fm' NOT FOUND RC=28
DMSTPE003E INVALID OPTION 'option'  RC=24
DMSTPE010E PREMATURE EOF ON FILE 'fn ft fm' RC=40
DMSTPE014E INVALID FUNCTION 'function'  RC=24
DMSTPE017E INVALID DEVICE ADDRESS 'cuu'  RC=24
DMSTPE023E NO FILETYPE SPECIFIED  RC=24
DMSTPE027E INVALID DEVICE 'device name'  RC=24
DMSTPE029E INVALID PARAMETER  'parameter' IN  THE OPTION  'option' FIELD
           RC=24
DMSTPE037E DISK 'mode' IS READ/ONLY  RC=36
DMSTPE042E NO FILEID SPECIFIED RC=24
DMSTPE043E 'TAPn(cuu)' IS FILE PROTECTED  RC=36
DMSTPE047E NO FUNCTION SPECIFIED  RC=24
DMSTPE048E INVALID MODE 'mode'  RC=24
DMSTPE057E INVALID RECORD FORMAT  RC=32
DMSTPE058E END-OF-FILE OR END-OF-TAPE  RC=40
DMSTPE070E INVALID PARAMETER 'parameter'  RC=24
DMSTPE096E FILE 'fn ft' DATA BLOCK COUNT INCORRECT  RC=32
DMSTPE104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSTPE105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSTPE110S ERROR READING 'TAPn(cuu)'  RC=100
DMSTPE111S ERROR WRITING 'TAPn(cuu)'  RC=100
DMSTPE113S TAPn(cuu) NOT ATTACHED  RC=100
DMSTPE115S {CONVERSION|{7|9}-TRACK|{800|6250} BPI|TRANSLATION|DUAL
           DENSITY} FEATURE NOT SUPPORTED ON DEVICE 'cuu'  RC=88
DMSTPE431E 'TAPn(cuu)' VOL1 LABEL MISSING  RC=32
```

TAPEMAC

Use the TAPEMAC command to create a CMS MACLIB from an unloaded partitioned data set (PDS) from a tape created by the IEHMOVE utility program under OS. The PDS from which the tape was created can be blocked, but the logical record length must be 80. The format of the TAPEMAC command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│             │    ┌                            ┐                          │
│  TAPEMAC    │ fn │SL  [labeldefid]            │    [(options[)]]         │
│             │    │NSL filename  [ID=identifier]│                         │
│             │    └                            ┘                          │
│             │                          options:                          │
│             │                          ┌    ┐┌            ┐              │
│             │                          │TAPn││ITEMCT yyyyy│              │
│             │                          │TAP1││ITEMCT 50000│              │
│             │                          └    ┘└            ┘              │
└──────────────────────────────────────────────────────────────────────────┘
```

where:

fn specifies the filename of the first, or only, CMS MACLIB to be created on the A-disk. If fn MACLIB already exists on the A-disk, the old one is erased; no warning message is issued.

SL means that the tape has standard labels. The default is SL without a labeldefid. With the default specification, the standard header labels are only displayed on the user's terminal. If labeldefid is specified, the standard labels are not displayed, but are checked by the tape label checking routine.

NSL means that the tape has nonstandard labels.

labeldefid

identifies the LABELDEF command that supplies descriptive label information for the file tc be processed. The labeldefid given here must match the 1- to 8-character identifier specified as the filename on the LABELDEF command that was previously issued.

filename is the CMS filename of a routine to process nonstandard labels. The filetype must be TEXT or MODULE. If both TEXT and MODULE files exist, the MODULE file is used. MODULE files that are used for NSL routines with the TAPEMAC command must be created so that they start at an address above X'21000'. This prevents the NSL modules from overlaying the command. See the section "Tape Labels in CMS" in the VM/370 CMS User's Guide for details on how to write routines to process nonstandard labels.

ID=identifier

specifies a 1- to 8-character identifier to be passed to a user-written NSL routine. You may use the identifier in any way you want to identify the file being processed. The identifier is passed to the user routine exactly as specified in the ID operand. If an identifier is not specified, blanks are passed. See the section "Tape Labels in CMS" in the VM/370 CMS User's Guide for details on communicating with routines that process nonstandard labels.

Options:

TAPn specifies the symbolic address of the tape, where n is a number between 1 and 4 corresponding to virtual device addresses 181 through 184, respectively. The default is TAP1.

ITEMCT yyyyy
specifies the item count threshold of each MACLIB to be created, which is the maximum number of records to be written into each file. yyyyy is a number between 0 and 62500 (commas are not allowed). If ITEMCT is not specified, the default is 50000.

Usage Notes

1. Tape records are read and placed into fn MACLIB until the file size exceeds the ITEMCT (item count); loading then continues until the end of the current member is reached. Then another CMS file is created; its filename consists of the number 2 appended to the end of the filename specified (fn) if the filename is seven characters or less. The appended number overlays the last character of the filename if the name is eight characters long. Loading then continues with this new name. For example, if you enter the command:

   ```
   tapemac mylib
   ```

   you may create files named MYLIB MACLIB, MYLIB2 MACLIB, MYLIB3 MACLIB, and so on.

   This process continues until up to nine CMS files have been created. If more data exists on the tape than can fit in nine CMS files, processing is terminated with the error message DMSTMA139S. The maximum size of the unloaded PDS which can be loaded into CMS MACLIBs would be approximately 9 times 62500 or 584,500 records.

2. Only header labels of the first file encountered are displayed or checked if SL or SL labdefid is specified. Trailer labels are not processed or displayed; they are skipped.

3. The following examples illustrate the different ways tape labels are processed by TAPEMAC. The command

   ```
   tapemac mac6 sl
   ```

   displays any standard VOL1 or HDR1 labels on a tape before loading maclib MAC6. It does not stop before loading the MACLIB.

   If you specify

   ```
   labeldef taplab fid macfile crdte 77106
   tapemac mac8 sl taplab
   ```

   CMS checks the HDR1 label on the tape before loading MAC8. It uses the information you supplied in the LABELDEF command TAPLAB to check the label. If there are discrepancies between fields you specified in the LABELDEF command and in the actual tape label, the MACLIB is not loaded.

   If you specify

   ```
   tapemac mac10 nsl nsl3
   ```

   CMS uses your own routine NSL3 to process tape labels before loading MAC10.

Responses

The TAPEMAC command displays the message:

```
LOADING fn MACLIB
```

for each macro library created.

Other Messages and Return Codes

```
DMSTMA001E NO FILENAME SPECIFIED  RC=24
DMSTMA003E INVALID OPTION 'option'  RC=24
DMSTMA057E INVALID RECORD FORMAT  RC=32
DMSTMA070E INVALID PARAMETER 'parameter'  RC=24
DMSTMA105S ERROR nn WRITING FILE fn ft ON DISK  RC=100
DMSTMA109S VIRTUAL STORAGE CAPACITY EXCEEDED  RC=104
DMSTMA110S ERROR READING TAPn  RC=100
DMSTMA137S ERROR nn ON STATE FOR fn ft  RC=100
DMSTMA138S ERROR nn ERASING 'fn ft' BEFORE LOADING TAPE  RC=100
DMSTMA139S TAPE FILE EXCEEDS 9 CMS MACLIBS  RC=104
DMSTMA420E NSL EXIT FILENAME MISSING OR INVALID  RC=24
```

TAPPDS

Use the TAPPDS command to create CMS disk files from tapes that are used as input to or output from the following OS utility programs:

- IEBPTPCH -- tape files must be the result of an IEBPTPCH punch operation from either a sequential or partitioned data set in OS. The default attributes (IEBPTPCH DCB) must have been issued:

  ```
  DCB=(RECFM=FA,LRECL=81,BLKSIZE=81)
  ```

  -- tape files may be blocked or unblocked and must be in the format accepted by IEBUPDTE as "control data set" (SYSIN) input with a control statement

  ```
  ./ ADD...
  ```

  preceding the records to be placed in each partitioned data set member (OS) or separate CMS file (CMS)).

- IEBUPDTE -- tape files may be blocked or unblccked.

- IEHMOVE -- unloaded partitioned data sets are read.

The tape can contain OS standard labels or be unlabeled. The format of the TAPPDS command is:

```
┌────────────────────────────────────────────────────────────────────────────┐
│          │  ┌    ┌    ┌   ┐┐┐     ┌                                 ┐   │
│ TAPPDS   │  │fn  │ft  │fm│││     │SL [labeldefid]                  │   │
│          │  │ *  │ *  │A1│││     │NSL filename  [ID=identifier]│   │
│          │  │    │    │* │││     └                                 ┘   │
│          │  └    └    └   ┘┘┘     [(options[)]]                          │
│          │                ┌       ┐  ┌       ┐    ┌    ┐               │
│          │  options:      │PDS    │  │COL1   │    │TAPn│               │
│          │                │NOPDS  │  │NOCOL1│    │TAP1│               │
│          │                │UPDATE│  └       ┘    └    ┘               │
│          │                └       ┘                                    │
│          │                ┌       ┐  ┌         ┐                       │
│          │                │ END   │  │MAXTEN   │                       │
│          │                │ NOEND│  │NOMAXTEN│                       │
│          │                └       ┘  └         ┘                       │
└────────────────────────────────────────────────────────────────────────────┘
```

where:

fn is the filename of the disk file to be created from the sequential tape file. If the tape contains members of a partitioned data set (PDS), fn must be specified as an asterisk (*); one file is created for each member with a filename the same as the member name. If NOPDS or UPDATE is specified and you do not specify fn or specify it as an asterisk (*), the default filename is TAPPDS.

ft is the filetype of the newly created files. The default filetypes are CMSUT1 (for PDS or NOPDS) and ASSEMBLE (for UPDATE). The defaults are used if ft is omitted or specified as *.

fm is the mode of the disk to contain the new files. If this field is omitted or specified as an asterisk (*), A1 is assumed.

SL means that the tape has standard labels. The default is SL without a labeldefid. With the default specification, the standard labels are displayed on the user's terminal. If labeldefid is specified, the standard labels are not displayed, but are checked by the tape label checking routine.

NSL means that the tape has nonstandard labels.

labeldefid identifies the LABELDEF command, which supplies descriptive label information for the file to be processed. The labeldefid given here must match the 1- to 8-character specified as the filename on the LABELDEF command that was previously issued.

filename is the CMS filename of a routine to process nonstandard labels. The filetype must be TEXT or MODULE. If both TEXT and MODULE files exist, the MODULE file is used. MODULE files that are used for NSL routines with the TAPPDS command must be created so that they start at an address above X'21000'. This prevents the MODULE files from overlaying the command. See the section "Tape Labels in CMS" in the *VM/370 CMS Users's Guide* for details on writing routines to process nonstandard labels.

ID=identifier
specifies a 1- to 8-character identifier to a user-written NSL routine. You may use the identifier in any way you want to identify the file being processed. The identifier is passed to the user routine exactly as specified in the operand. If an identifier is not specified, blanks are passed. See the section "Tape Labels in CMS" in the *VM/370 CMS User's Guide* for details on communication with routines that process nonstandard labels.

Note: If either SL or NSL is specified for tape label processing, the fn, ft, and fm operands must all be specified. They may be specified by asterisks (*) if you want default values; however, none of the three operands may be omitted.

Options: If conflicting options are specified, the last one entered is the one that is used. All options, except TAPn, are ignored when unloaded (IEHMOVE) PDS tapes are read.

PDS indicates that the tape contains members of an OS partitioned data set, each preceded by a MEMBER NAME=name statement. The tape must have been created by the OS IEBPTPCH service program if this option is specified.

NOPDS indicates that the contents of the tape will be placed in one CMS file.

UPDATE indicates that the tape file is in IEBUPDTE control file format. The filename of each file is taken from the NAME= parameter in the "./ ADD" record that precedes each member. (See Usage Note 2.)

COL1 reads data from columns 1-80. You should specify this option when you use the UPDATE option.

NOCOL1 reads data from columns 2-81; column 1 contains control character information. This is the format produced by the OS IEBPTPCH service program.

TAPn is the tape unit number. n can be 1, 2, 3, or 4, representing virtual units 181, 182, 183, and 184, respectively. If not specified, TAP1 is assumed.

END considers an END statement (characters 'END ' in columns 2-5) a delimiter for the current member.

NOEND specifies that END statements are not to be treated as member delimiters, but are to be processed as text.

MAXTEN reads up to ten members. This is valid only if the PDS option is selected.

NOMAXTEN reads any number of members.

Usage Notes

1. You can use the TAPE command to position a tape at a particular tape file before reading it with the TAPPDS command. If the tape has OS standard labels, TAPDDS will read and display the "VOL1" and "HDR" records at the terminal. If the file you want to process is not at the beginning of the tape, the TAPE command must be used to position the tape at a particular tape file before reading it with the TAPPDS command. Be aware that each file on an OS standard label tape is actually three physical files (HDR, DATA, TRAILER). If positioning to other than the first file, the user must skip more physical tape files (3n-3 if positioning to the header labels, 3n-2 if positioning to the data file, where n is the number of the file on the tape).

2. If you use the UPDATE option, you must also specify the COL1 option. Each tape record is scanned for a "./ ADD" record beginning in column 1. When a "./ ADD" record is found, subsequent records are read onto disk until the next "./ ADD" record is encountered or until a "./ ENDUP" record is encountered.

A "./ ENDUP" record or a tape mark ends the TAPPDS command execution; the tape is not repositioned.

"./ label" records are not recognized by CMS and are included in the file as data records.

If the NAME= parameter is missing on the "./ ADD" record or if it is followed by a blank, TAPPDS uses the default filename, TAPPDS, for the CMS disk file. If this happens more than once during the execution of the command, only the last unnamed member is contained in the TAPPDS file.

3. If you are reading a macro library from a tape created by the IEHMOVE utility, you can create a CMS MACLIB file directly by using the TAPEMAC command.

4. Only header labels of the first file encountered are displayed or checked if SL or SL labeldefid is specified. Trailer labels are not processed or displayed; they are skipped. When more than one file is processed by one issuance of the TAPPDS command, only the first file has its standard labels processed. Standard labels are skipped on succeeding files.

5. The following examples illustrate different ways in which tape labels are processed by TAPPDS. If you specify

```
tappds fileg cmsut1 * sl
```

then, before loading the PDS into fileg, CMS displays a VOL1 and HDR1 label if it exists on the tape. It does not stop before the PDS is loaded; therefore, you cannot use the tape label to suppress loading if the wrong tape has been mounted.

If you specify

```
labeldef label2 fid pds1 volid xyz
tappds fileh cmsut1 * sl label2
```

CMS uses the label information specified to check the label on the tape before loading your PDS. If there are discrepancies, the PDS is not loaded.

If you specify

```
tappds filej * * nsl nonstd
```

CMS uses your own routine called NONSTD to process tape labels before loading the PDS.

<u>Responses</u>

DMSTPD703I FILE 'fn ft [fm]' COPIED

The named file is copied to disk.

DMSTPD707I TEN FILES COPIED

The MAXTEN option was specified and ten members have been copied.

<u>Note</u>: If the tape being read contains standard OS labels, the labels are displayed at the terminal.

Other Messages and Return Codes

```
DMSTPD003E INVALID OPTION 'option'  RC=24
DMSTPD058E END-OF-FILE OR END-OF-TAPE  RC=40
DMSTPD105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSTPD109S VIRTUAL STORAGE CAPACITY EXCEEDED  RC=104
DMSTPD110S ERROR 'nn' READING 'TAPn(cuu)'  RC=100
DMSTPD420E NSL EXIT FILENAME MISSING OR INVALID RC=24
```

TXTLIB

Use the TXTLIB command to update CMS text libraries. The format of the TXTLIB command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
| TXTlib   | ( GEN libname fn1 [fn2 ...]                      )            |
|          |                                                  |            |
|          | ) ADD libname fn1 [fn2 ...]                      (  options:  |
|          |                                                  |   ┌     ┐  |
|          | ) DEL libname membername1 [membername2...]       (   |TERM |  |
|          |                                                  |   |DISK |  |
|          | ( MAP libname  [(options...[)]]                  )   |PRINT|  |
|          |                                                      └     ┘  |
└──────────────────────────────────────────────────────────────────────────┘
```

where:

GEN creates a TXTLIB on your A-disk. If a TXTLIB with the same name already exists, it is replaced.

ADD adds TEXT files to the end of an existing TXTLIB on a read/write disk. No checking is done for duplicate names, entry points, or CSECTs.

DEL deletes members from a TXTLIB on a read/write disk and compresses the TXTLIB to remove unused space. If more than one member exists with the same name, only the first entry is deleted.

MAP lists the names (entry points) of TXTLIB members, their locations in the library, and the number of entries.

libname specifies the filename of a file with a filetype of TXTLIB, which is to be created or listed or from which members are to be deleted or added.

fn1 [fn2...]
specifies the name(s) of file(s) with filetype(s) of TEXT, that you want to add to a TXTLIB.

membername1 [membername2...]
specifies the name(s) of TXTLIB member(s) that you want to delete.

Options:

TERM displays information about the TXTLIB on your terminal.

DISK writes a CMS file, named libname MAP A5, that contains a list of TXTLIB members.

PRINT spools a copy of the TXTLIB map to the virtual printer.

Usage Notes

1. When a TEXT file is added to a library, its membername(s) are taken from the CSECT names or NAME statements in the TEXT file. Deletions and LOAD or INCLUDE command references must be made on these names. For example, a TEXT file with a filename of TESTPROG that contains CSECTs named CHECK and RECHECK, when added to a TXTLIB, creates members named CHECK and RECHECK.

2. Members must be deleted by their initial entry in the dictionary (that is, their "name" or the first ID name). Any attempt to delete a specific alias or entry point within a member will result in a "Not found" message.

3. If you want your TXTLIBs to be searched for missing subroutines during CMS loader processing; you must identify the TXTLIB on a GLOBAL ccmmand; for example:

   ```
   global txtlib newlib
   ```

4. You may add OS linkage editor control statements NAME, ALIAS, ENTRY, and SETSSI to a TEXT file before adding it to a TXTLIB. Ycu must follow OS linkage editor conventions concerning format (column 1 must be blank) and placement within the TEXT file. The specified entry point must be located within the CSECT.

5. TXTLIB members are not fully link-edited, and may return erroneous entry points during dynamic loading.

6. The total number of members in the TXTLIB file cannot exceed 1000. When this number is reached, an error message is displayed. The total number of entry points in a member cannot exceed 255. When this number is reached, an error message is displayed and the next text file (if there is one) is processed. The text litrary created includes all the text files entered up to (but not including) the one that caused the overflow.

7. TERM or PRINT options will erase the old MAP file, if one exists.

<u>Responses</u>

When the TXTLIB MAP command is issued with the TERM option, the contents of the directory of the specified text library are displayed at the terminal. The number of entries in the text library (xxx) is also displayed.

```
ENTRY INDEX
name  location
  .     .
  .     .
  .     .
xxx ENTRIES IN LIBRARY
```

<u>Other Messages and Return Codes</u>

```
DMSLBT001E NO FILENAME SPECIFIED  RC=24
DMSLBT002E FILE 'fn ft' NOT FOUND RC=28
DMSLBT002W FILE 'fn ft' NOT FOUND RC=4
DMSLBT003E INVALID OPTION 'option'  RC=24
DMSLBT013E MEMBER 'name' NOT FOUND IN LIBRARY 'fn ft fm'  RC=32
DMSLBT014E INVALID FUNCTION 'function'  RC=24
DMSLBT037E DISK 'mode' is READ/ONLY  RC=36
DMSLBT046E NO LIBRARY NAME SPECIFIED  RC=24
DMSLBT047E NO FUNCTION SPECIFIED  RC=24
DMSLBT056E FILE  'fn  ft  fm'  CONTAINS  [NAME|ALIAS|ENTRY|ESD]  INVALID
           RECORD FORMATS  RC=32
DMSLBT056W FILE  'fn ft  fm'  CONTAINS [{NAME|ALIAS|ENTRY|ESD}]  INVALID
           RECORD FORMATS  RC=4
DMSLBT069E DISK 'mode' NOT ACCESSED  RC=36
DMSLBT104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSLBT105S ERROR 'nn' WRITING FILE 'fn ft fm' ON CISK  RC=100
DMSLBT106S NUMBER OF MEMBER NAMES EXCEEDS MAX  'nnnn'.  FILE 'fn ft' NCT
           ADDED  RC=88
DMSLBT213W LIBRARY 'fn ft fm' NOT CREATED  RC=4
```

TYPE

Use the TYPE command to display all or part of a CMS file at the terminal in either EBCDIC or the hexadecimal representation of the EBCDIC code. The format of the TYPE command is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│      │                                                                   │
│ Type │ fn ft [fm]  ┌ rec1 ┌recn┐┐ [(options...[)] ]                        │
│      │        *    │  *   │ *  ││                                         │
│      │             │  1   │    ││                                         │
│      │             └      └    ┘┘                                         │
│      │                                                                   │
│      │         options:                                                  │
│      │                   ┌          ┌      ┐┐    ┌               ┐        │
│      │          [HEX]    │COL {xxxxx}-│yyyyy││    │MEMBER {  *  }│        │
│      │                   │    {  1  } │lrecl││    │       { name}│        │
│      │                   └          └      ┘┘    └               ┘        │
└──────────────────────────────────────────────────────────────────────────┘
```

where:

fn is the filename of the file to be displayed.

ft is the filetype of the file to be displayed.

fm is the filemode of the file to be displayed. If this field is omitted, the A-disk and its extensions are searched to locate the file. If fm is specified as an asterisk (*), all disks are searched, and the first file found is displayed.

rec1 is the record number of the first record to be displayed. This field cannot contain special characters. If rec1 is greater than the number of records in the file, an error message is displayed. If this field is omitted or entered as an asterisk (*), a record number of 1 is assumed.

recn is the record number of the last record to be displayed. This value cannot contain embedded commas. If this field is nct specified, is entered as an asterisk (*), or is greater than the number of records in the file, displaying continues until end cf file is reached.

Options:

COL xxxxx-yyyyy
displays only certain columns of each record. xxxxx specifies the start column and yyyyy the end column of the field within the record that is to be displayed. The string xxxxx-yyyyy may have a maximum of eight characters; additional characters are truncated.

If columns are not specified, the entire record is displayed unless the filetype is LISTING, in which case the first position of each record is not displayed, since it is assumed to be a carriage control character.

HEX displays the file in hexadecimal format.

```
MEMBER ⎧ *    ⎫
MEM    ⎩ name ⎭
```

displays member(s) of a library. If ft is MACLIB or TXTLIB, a MEMBER entry can be specified. If an asterisk (*) is specified, all members of the library are displayed. If a name is specified, only that particular member is displayed.

Usage Notes

1. If the HEX option is specified, each record can be displayed in its entirety; if not, no more than 130 characters of each record can be displayed.

2. The length of each output line is limited to 130 characters or the current terminal linesize (as specified by the CP TERMINAL command), whichever is smaller.

Responses

The file is displayed at the terminal according to the given specifications. When you use the HEX option, each record is preceded by a header record:

```
RECORD nnnnnnnnnn LENGTH=nnnnnnnnnn
```

Other Messages and Return Codes

```
DMSTYP002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSTYP003E INVALID OPTION 'option'  RC=24
DMSTYP005E NO 'option' SPECIFIED  RC=24
DMSTYP009E COLUMN 'col' EXCEEDS RECORD LENGTH  RC=24
DMSTYP013E MEMBER 'name' NOT FOUND IN LIBRARY  RC=32
DMSTYP029E INVALID PARAMETER 'parameter' [IN  THE OPTION 'option' FIELD]
           RC=24
DMSTYP033E FILE 'fn ft fm' IS NOT A LIBRARY  RC=32
DMSTYP039E NO ENTRIES IN LIBRARY 'fn ft fm'  RC=32
DMSTYP049E INVALID LINE NUMBER 'line number'  RC=24
DMSTYP054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSTYP062E INVALID * IN FILEID  RC=20
DMSTYP104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
```

UPDATE

Use the UPDATE command to modify program source files. The UPDATE command accepts a source input file and one or more files containing UPDATE control statements and updated source records; then it creates an updated source output file, an update log file indicating what changes, if any, were made, and an update record file if more than a single update file is applied to the input file. The format of the UPDATE command is:

```
┌─────────────────────────────────────────────────────────────────────────┐
│        │                                                                │
│ Update │ fn1 ┌          ┌                    ┐┐                         │
│        │     │ft1       │fm1 [fn2 [ft2 [fm2]]]││ [(options...[)]]       │
│        │     │ASSEMBLE  │A1                  ││                         │
│        │     └          └                    ┘┘                         │
│        │                                                                │
│        │                ┌      ┐   ┌       ┐   ┌     ┐   ┌      ┐       │
│        │     options:   │REP   │   │SEQ8   │   │INC  │   │CTL   │       │
│        │                │NOREP │   │NOSEQ8 │   │NOINC│   │NOCTL │       │
│        │                └      ┘   └       ┘   └     ┘   └      ┘       │
│        │                                                                │
│        │                ┌      ┐   ┌       ┐   ┌     ┐   ┌      ┐       │
│        │                │STK   │   │TERM   │   │DISK │   │STOR  │       │
│        │                │NOSTK │   │NOTERM │   │PRINT│   │NOSTOR│       │
│        │                └      ┘   └       ┘   └     ┘   └      ┘       │
└─────────────────────────────────────────────────────────────────────────┘
```

where:

fn1 ft1 fm1
: is the file identifier of the source input file. The file must consist of 80-character card image records with sequence fields in positions 73 through 80 or 76 through 80. If the filetype or filemode are omitted, ASSEMBLE and A1 are assumed, respectively.

fn2 ft2 fm2
: is the file identifier of the update file. If the NOCTL option is in effect, this file must contain UPDATE control statements and updated source records. The default file identifier is fn1 UPDATE A1. If the CTL option is specified, this file must be a control file that lists the update files to be applied; the default file identifier is fn1 CNTRL A1.

Options:

REP
: creates an output source file with the same filename as the input source file. If the output file is placed on the same disk as the input file, the input file is erased.

NOREP
: retains the old file in its original form, and assigns a different filename to the new file, consisting of a dollar sign ($) plus the first seven characters of the input filename (fn1).

SEQ8
: specifies that the entire sequence field (columns 73 through 80) contains an eight-digit sequence number on every record of source input.

NOSEQ8 specifies that columns 73-75 contain a three-character label field, and that the sequence number is a five-digit value in columns 76-80.

Note: Source files sequenced by the CMS editor are sequenced, by default, with five-digit sequence numbers.

INC increments sequence numbers in cclumns 73 through 80 in each record inserted into the updated output file, according to specifications in UPCATE control statements.

NOINC puts asterisks (********) in the sequence number field of each updated record inserted from the update file.

CTL specifies that fn2, ft2, and fm2 describe an update control file for applying multiple update files to the source input file. (See "The CTL Option.")

Note: The CTL option implies the INC option.

NOCTL specifies that a single update file is to be applied to the source input file.

STK stacks information from the ccntrol file in the CMS console stack. STK is valid only if the CTL option is also specified and is useful only when the UPDATE command is executed in an EXEC procedure.

NOSTK does not stack control file infcrmation in the console stack.

TERM displays warning messages at the terminal whenever a sequence or update control card error is discovered. (Such warning messages appear in the update log, whether they are displayed at the terminal or not.)

NOTERM suppresses the display of warning messages at the terminal. However, error messages that terminate the entire update procedure are displayed at the terminal.

DISK places the update log file on disk. This file has a file identifier "fn UPDLOG", where "fn" is the filename of the file being updated.

PRINT prints the update log file directly on the virtual printer.

STOR specifies that the source input file is to be read into storage and the updates performed in storage prior to placing the updated source file cn disk. This option is meaningful only when used with the CTL option since the benefit of increased processing speed is realized when processing multiple updates. STCR is the default when CTL is specified.

NOSTOR specifies that no updating is to take place in storage. NOSTOR is the default when single updates are being applied (CTL is omitted from the command line).

UPDATE CONTROL STATEMENTS

The UPDATE control statements let you insert, delete, and replace source records, as well as resequence the output file.

All references to the sequence field of an input record refer to the numeric data in columns 73-80 of the source record, or columns 76-80 if NOSEQ8 is specified. Leading zeros in sequence fields are not required. If no sequence numbers exist in an input file, a preliminary UPDATE with only the './ S' control statement can be used to establish file sequencing.

Sequence numbers are checked while updates are being applied; an error condition results if any sequence errors occur in the update control statements, and warnings are issued if an error is detected in the sequencing of the input file. Any source input records with a sequence field of eight blanks are skipped, without any indication of a sequence error. Such records may be replaced or deleted only if they occur within a range of records that are being replaced or deleted entirely and if that range has limits with valid sequence numbers. There is no means provided for specifying a sequence field of blanks on an UPDATE control statement.

Control Statement Formats

All UPDATE control statements are identified by the characters './' in columns 1 and 2 of the 80-byte record, followed by one or more blanks and additional, blank-delimited fields. Control statement data must not extend beyond column 50.

SEQUENCE Control Statement -- resequences the updated source output file in columns 73-80 (if SEQ8 is specified), or in columns 76-80 with the label placed in columns 73-75 (if NOSEQ8 is specified). The format of the SEQUENCE control statement is:

```
./ S [seqstrt [seqincr [label]]]
```

where:

seqstrt — is a one- to eight-digit numeric field specifying the first decimal sequence number to be used. The default value is 1000 if SEQ8 is specified and 10 if NOSEQ8 is specified.

seqincr — is a one- to eight-digit numeric field specifying the decimal increment for resequencing the output file. The default is the "seqstrt" value.

label — is a three-character field to be duplicated in columns 73-75 of each source record if NOSEQ8 is specified. The default value is the first three characters of the input filename (fn1).

If you use the SEQUENCE statement, it must be the first statement in the update file. If any valid control statement precedes it, the resequence operation is suppressed.

Each source record is resequenced in columns 73-80 as it is written onto the output file, including unchanged records from the source file and records inserted from the update file.

INSERT Control Statement -- inserts all records following it, up to the next control statement, into the output file. The format of the INSERT control statement is:

```
./ I  seqno  [$  [seqstrt  [seqincr]]]
```

where:

seqno is the sequence number of the record in the source input file following which new records are to be added.

$ is an optional delimiter indicating that the inserted records are to be sequenced by increments.

seqstrt is a one- to eight-digit numeric field specifying the first decimal number to be used for sequencing the inserted records.

seqincr is a one- to eight-digit numeric field specifying the decimal increment for sequencing the inserted records.

All records following the "./ I" statement, up to the next control statement, are inserted in the output file following the record identified by the "seqno" field. If the NOINC option is specified, each inserted record is identified with asterisks (********) in columns 73-80. If either the INC or CTL option is specified, the records are inserted unchanged in the output file, or they are sequenced according to the "seqstrt" and "seqincr" fields, if the dollar sign ($) key is specified.

The default sequence increment, if the dollar sign is included, is determined by using one tenth of the least significant, nonzero digit in the seqno field, with a maximum of 100. The default seqstrt is computed as seqno plus the default seqincr. For example, the control statement:

```
./ I  2600 $ 2610
```

causes the inserted records to be sequenced XXX02610, XXX02620, and so forth (NOSEQ8 assumed here). For the control statement:

```
./ I  240000 $
```

the defaulted seqincr is the maximum, 100, and the starting sequence number is 240100. SEQ8 is assumed, so the inserted records are sequenced 00240100, 00240200, and so forth.

If either INC or CTL is specified but the dollar sign is not included, whatever sequence number appears on the inserted records in the update file is included in the output file.

DELETE Control Statement -- deletes one or more records from the source file. The format of the DELETE control statement is:

```
./ D  seqno1  [seqno2]  [$]
```

where:

seqno1 is the sequence number identifying the first or only record to be deleted.

seqno2 is the sequence number of the last record to be deleted.

$ is an optional delimiter indicating the end of the control fields.

All records of the input file, beginning at seqno1, up to and including the seqno2 record, are deleted from the output file. If the seqno2 field is omitted, only a single record is deleted.

REPLACE Control Statement -- replaces one or more input records with updated records from the update file. The format of the REPLACE control statement is:

```
./  R seqno1 [seqno2] [$ [seqstrt [seqincr]]]
```

where:

seqno1 is the sequence number of the first input record to be replaced.

seqno2 is the sequence number of the last record to be replaced.

$ is an optional delimiter key indicating that the substituted records are to be sequenced incrementally.

seqstrt is a one- to eight-digit numeric field specifying the first decimal number to be used for sequencing the substituted records.

seqincr is a one- to eight-digit numeric field specifying the decimal increment for sequencing the substituted records.

All records of the input file, beginning with the seqno1 record, up to and including the seqno2 record, are replaced in the output file by the records fcllowing the "./ R" statement in the update file, up to the next control statement. As with the "./ D" (delete) function, if the seqno2 field is omitted, only a single record is replaced, but it may be replaced by more than a single inserted record. The "./ R" (replace) function is performed as a delete followed by an insert: thus, the number of statements inserted need not match the number deleted. The dollar sign ($), seqstrt, and seqincr processing is identical to that for the insert function.

COMMENT Statement --allows inserting supplemental information that the user may want. The format of the COMMENT statement is:

```
./  *  [comment]
```

where:

* indicates that this is a comment statement and is only copied into the update log file.

SUMMARY OF FILES USED BY THE UPDATE COMMAND

The following discussion shows input and output files used by the UPDATE command for a:

- Single-level update
- Multilevel update
- Multilevel update with an auxiliary control file

Disk Mode of Output Files: If several read/write disks are accessed when the UPDATE command is invoked, the following steps are taken to determine the disk upon which the output files are to be placed (the search stops as soon as one of the following steps is successful):

1. If the disk on which the original source file resides is read/write, then the output files are placed on that disk.

2. If that disk is a read-only extension of a read/write disk, then the output files are placed on that particular read/write disk.

3. If neither of the other steps is successful, the output files are placed on the primary read/write disk (the A-disk).

Single-Level Update

```
┌─────────┐
│         │  fn ASSEMBLE          ┌─────────┐
│         │  fn UPDATE            │         │  $fn ASSEMBLE
│         │                       │         │  fn UPDLOG
└─────────┘                       │         │
                                  └─────────┘
                update fn
```

Notes:

fn ASSEMBLE is the source input file.

fn UPDATE contains UPDATE control statements and updated source input records.

$fn ASSEMBLE is the updated source file, incorporating changes, additions, and deletions specified in the update file. The output filetype is always the same as the filetype of the input file. These default filetypes and filemodes can be overridden on the command line; for example:

```
     update testprog cobol b fix cobol b (rep
```

results in a source file TESTPROG COBOL B being updated with contrcl statements contained in the file FIX COBOL B. The output file replaces the existing TESTPROG COBOL B.

fn UPDLOG contains a record of updates applied. If you do not want this file written cn disk, specify the PRINT option.

Multilevel Update

```
┌─────────┐
│         │  fn ASSEMBLE          ┌─────────┐
│         │  fn CNTRL             │         │  $fn ASSEMBLE
│         │  fn UPDTABC           │         │  fn UPDLOG
│         │  fn UPDTXYZ           │         │  fn UPCATES
│         │                       │         │
└─────────┘                       └─────────┘

             update fn (ctl
```

Notes:

fn ASSEMBLE is the source input file.

fn CNTRL is the control file that lists updates to be applied to the source file. These default filetypes and filemodes can be overridden cn the command line; for example:

```
     update acct pliopt a test cntrl a (ctl
```

results in the file TEST CNTRL being used by the UPDATE command to locate the update files for ACCT PLIOPT.

fn UPDTABC *and* *fn* *UPDTXYZ* are update files containing UPDATE control statements and new source records. These files must have filenames that are the same as the source input file. The first four characters of the filetype must be "UPDT." The UPDATE command searches all accessed disks to locate the update files.

$fn *ASSEMBLE* is the updated source file, incorporating changes, additions, and deletions specified in the update files. The filetype is always the same as the filetype of the source input file.

fn UPDLOG contains a record of updates applied. If you do not want this file written on disk, specify the PRINT option.

fn UPDATES summarizes the updates applied to the source file.

The CONTROL FILE (fn CNTRL) may not contain UPDATE control statements. It may only list the filetypes of the files that contain UPDATE control statements. This control file contains the records:

```
TEXT MACS CMSLIB
TWO UPDTABC
ONE UPDTXYZ
```

where *UPDTABC* and *UPDTXYZ* are the filetypes of the update files. The UPDATE command applies these updates to the source file beginning with the last record in the control file. Thus, the updates in fn UPDTXYZ are applied before the updates in fn UPDTABC.

When you create update files whose filetypes begin with 'UPDT', you may omit these characters when you list the updates in the control file; thus, the CNTRL file may be written:

```
TEXT MACS CMSLIB
TWO ABC
ONE XYZ
```

TEXT, *TWO*, *ONE*: The first column of the control file consists of an update level identifier, which may be from one to five characters long. These identifiers are used by VM/370 updating procedures, like the VMFASM EXEC, to locate and identify text decks produced by multilevel updates.

MACS: The first record in the control file must be a MACS record which contains an update level identifier (TEXT) and, optionally, lists up to eight macro library (MACLIB) filenames.

The information provided in the MACS card and the update level identifier are not used by the UPDATE command unless the STK option is specified. They are, however, required in the CNTRL file.

Multilevel Update with Auxiliary Control File

```
┌─────────┐
│         │  fn ASSEMBLE          ┌──────────┐   $fn ASSEMBLE
│         │  fn CNTRL             │          │   fn UPDLOG
│         │  fn UPDTABC           │          │   fn UPDATES
│         │  fn UPDTXYZ           │          │
│         │  fn AUXLIST           │          │
└─────────┘  fn FIX01             └──────────┘
             fn FIX02
```

```
update fn (ctl
```

Notes:

fn ASSEMBLE, fn CNTRL, fn UPDTABC, fn UPDTXYZ, $fn ASSEMBLE, fn UPDLOG, and fn UPDATES are used as described above, for "Multilevel Update," except that the CNTRL file contains:

```
TEXT MACS CMSLIB
TWO UPDTABC
ONE UPDTXYZ
TEXT AUXLIST
```

AUX in the filetype AUXLIST indicates that this is the filetype of an auxiliary control file that contains an additional list of updates. The first three characters of the filetype of an auxiliary control file must be "AUX"; the remaining character(s) (to a maximum of 5) may be anything. The filename must be the same as the source input file.

An auxiliary file may also be specified as:

```
xxxxx AUX
```

in the control file. For example, the record:

```
FIX TEST AUX
```

identifies the auxiliary file fn AUXTEST.

Note that if you give an auxiliary control file the filetype AUXPTF, the UPDATE command assumes that it is a simple update file and does nct treat it as an auxiliary file.

PREFERRED AUX FILE: A preferred AUX file may be specified. A preferred AUX file contains the version of an update that applies to your version of the source file. (There may be more than one version of the same update if there is more than one version of the source file. Fcr example, you need one version for the source file that has a system extension program product installed, and you need another version for the source file that does not have a program product installed.)

When you specify an auxiliary control file, you can specify more than one filetype. The first filetype indicates a file that UPDATE uses only on one condition: the files that the second and subsequent filetypes indicate do nct exist. If they do exist, this AUX file entry is ignored and no updating is done. The files that the second and subsequent filetypes indicate are preferred because, if they exist, UPDATE does nct use the file that the first filetype indicates. For example, assume that the file 'fn ASSEMBLE' does exist. The control file MYMODS CNTRL:

```
     TEXT  MACS  MYMACS  CMSLIB  OSMACRO

     MY2   AUXTEST

     MY1   AUXMINE AUXTEST
```

and the command:

```
     UPDATE fn ASSEMBLE * MYMODS CNTRL (CTL
```

would result in UPDATE finding the preferred auxiliary control file 'fn AUXTEST', and therefore not using 'fn AUXMINE' to update 'fn ASSEMBLE'. UPDATE would then proceed to the MY2 AUXTEST entry and update 'fn ASSEMBLE' with the updates listed in 'fn AUXTEST.' It is assumed that AUXTEST and AUXMINE list similar but mutually exclusive updates.

The search for a "preferred" auxfile will continue until one is found cr until the token is an invalid filetype; that is, less than four or more than eight characters. This token and the remainder of the line are considered a comment.

fn FIX01 and fn FIX02 are update files containing UPDATE control statements and new source records to be incorporated into the input file. When update files are listed in an auxiliary control file, they can have any filetype you choose but the filename must be the same as the source input file. The update files, as well as the AUX file, may be on any accessed disk. These are indicated in fn AUXLIST as follows:

```
     FIX02
     FIX01
```

The updates are applied from the bottom of the auxiliary file. Thus, fn FIX01 is applied to the source file before fn FIX02. Since the auxiliary file is listed at the bottom of the control file, these updates are applied before UPDTXYZ and UPDTABC.

ADDITIONAL CONTROL FILE RECORDS: In addition to the MACS record, the filetypes of update (UPDT) files, and the filetypes of auxiliary control (AUX) files, a control file may also contain:

- Comments. These records begin with an asterisk (*) in column 1. Comments are also valid in AUX files.

- PTF records. If the characters PTF appear in the update level identifier field, the UPDATE commanu expects the second field to contain the filetype of an update file. The filetype may be anything; the filename must be the same as the source input file.

- Update level identifiers not associated with update files.

The following example of a control file shows all the valid types of records:

```
     * Example of a control file
     ABC MACS MYLIB
     TEXT
     004  UPDTABC
     003  XYZ
     002  AUXLIST1
     001  LIST2  AUX
     PTF  TESTFIX
```

THE STK OPTION: The STK (stack) option is valid only with the CTL opticn and is meaningful only when the UPDATE command is invoked within an EXEC procedure.

When the STK option is specified, UPDATE stacks the following data lines in the console stack:

```
first line:  * update level identifier
second line: * library list from MACS record
```

The update level identifier is the identifier of the most recent update file that was found and applied. For example, if a control file contains

```
TEXT MACS CMSLIB OSMACRO TESTMAC
OFA  UPDTOFA
PFA  UPDTOFA
```

and the UPDATE command appears in an EXEC as follows:

```
UPDATE SAMPLE (CTL STK
&READ VARS &STAR &TEXT
&READ VARS &STAR &LIB1 &LIB2 &LIB3 &LIB4
```

then the variable symbols set by the &READ VARS statements have the following values if the file SAMPLE UPDTOFA is found and applied to the file SAMPLE ASSEMBLE:

Symbol	Value
&STAR	*
&TEXT	OFA
&LIB1	CMSLIB
&LIB2	OSMACRO
&LIB3	TESTMAC
&LIB4	null

The library list may be useful to establish macro libraries in a subsequent GLOBAL command within the EXEC procedure. If no update files are found, UPDATE stacks the update level identifier on the MACS record.

Responses

FILE 'fn ft fm,' REC #n = update control statement

This message is displayed when the TERM option is specified and an error is detected in an update file. It identifies the file and record number where the error is found.

DMSUPD177I WARNING MESSAGES ISSUED (SEVERITY=nn). ['REP' OPTICN IGNORED.]

Warning messages were issued during the updating process. The severity shown in the error message in the "nn" field is the highest of the return codes associated with the warning messages that were generated during the updating process.

The warning return codes have the following meanings:

RC = 4; Sequence errors were detected in the original source file being updated.

RC = 8; Sequence errors, which did not previously exist in the source file being updated, were introduced in the output file during the updating process.

RC = 12; Any other nonfatal error detected during the updating process. Such errors include invalid update file control statements and missing update or PTF files.

The severity value is passed back as the return code from the UPDATE command. In addition, if the REP option is specified in the command line, then it is ignored, and the updated source file has the fileid "$fn1 ft1", as if the REP option was not specified.

DMSUPD178I UPDATING ['fn ft fm'] WITH 'fn ft fm'

The specified update file is being applied to the source file. This message appears only if the CTL option is specified in the command line. The updating process continues.

DMSUPD304I UPDATE PROCESSING WILL BE DONE USING DISK

An insufficient amount of virtual storage was available to perform the updating in virtual storage, so a CMS disk must be used. This message is displayed only if NOSTOR was specified in the UPDATE command line.

Other Messages and Return Codes

```
DMSUPD001E NO FILENAME SPECIFIED  RC=4
DMSUPD002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSUPD003E INVALID OPTION 'option'  RC=24
DMSUPD007E FILE 'fn ft fm' IS NOT FIXED, 80 CHAR. RECORDS  RC=32
DMSUPD010W PREMATURE EOF OF FILE 'fn ft  fm' --SEQ NUMBER '........' NOT
           FOUND  RC=12
DMSUPD024E FILE 'UPDATE CMSUT1 fm' ALREADY EXISTS  RC=28
DMSUPD037E DISK 'mode' IS READ/ONLY  RC=36
DMSUPD048E INVALID MODE 'mode'  RC=24
DMSUPD065E 'option' OPTION SPECIFIED TWICE  RC=24
DMSUPD066E 'option' AND 'option' ARE CONFLICTING OPTIONS  RC=24
DMSUPD069E DISK 'mode' NOT ACCESSED  RC=36
DMSUPD070E INVALID PARAMETER 'parameter'  RC=24
DMSUPD104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSUPD105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSUPD174W SEQUENCE  ERROR  INTRODUCED  IN OUTPUT  FILE:  '........'  TO
           '........' RC=8
DMSUPD176W SEQUENCING OVERFLOW FOLLOWING SEQ NUMBER'........'  RC=8
DMSUPD179E MISSING OR DUPLICATE  'MACS' CARD IN CONTROL FILE  'fn ft fm'
           RC=32
DMSUPD180W MISSING PTF FILE 'fn ft fm'  RC=12
DMSUPD181E NO UPDATE FILES WERE FOUND  RC=40
DMSUPD182W SEQUENCE INCREMENT IS ZERO  RC=8
DMSUPD183E INVALID {CONTROL|AUX} FILE CONTROL CARD  RC=32
DMSUPD184W './S ' NOT FIRST CARD IN INPUT FILE --IGNORED  RC=12
DMSUPD185W INVALID CHAR IN SEQUENCE FIELD '........'  RC=12
DMSUPD186W SEQUENCE NUMBER '........' NOT FOUND  RC=12
DMSUPD187E OPTION 'STK' INVALID WITHOUT 'CTL'  RC=24
DMSUPD207W INVALID UPDATE FILE CONTROL CARD  RC=12
DMSUPD210W INPUT FILE SEQUENCE ERROR: '.........' TO '.........'  RC=4
DMSUPD299E INSUFFICIENT STORAGE TO COMPLETE UPDATE  RC=41
DMSUPD300E INSUFFICIENT STORAGE TO BEGIN UPDATE  RC=41
```

Immediate Commands

You can issue an Immediate command from the terminal only after causing an attention interruption by pressing the Attention key (or its equivalent). These commands are processed as soon as they are entered. The HT and RT Immediate commands are also recognized when they are stacked in an EXEC procedure, and the HT Immediate command can be appended to a CMS command preceded by a logical line end symbol (#). Any program execution in progress is suspended until the Immediate command is processed.

None of the Immediate commands issue responses.

HB

Use the HB command to stop the execution of a CMS batch virtual machine at the end of the current job. The format of the HB Immediate command is:

```
| HB |
```

Usage Notes

1. If the batch virtual machine is running disconnected, it must be reconnected.
2. When the HB command is executed, CMS sets a flag such that at the end of the current job, the batch processor generates accounting information for the current job and then logs off the CMS batch virtual machine.

HO

Use the HO command during the execution of a command or one of your programs to stop the recording of trace information. Program execution continues to its normal completion, and all recorded trace information is spooled to the printer. The format of the HO command is:

```
| HO |
```

HT

Use the HT command to suppress all terminal output generated by any CMS command or your program that is currently executing. The format of the HT command is:

```
| HT |                                                               |
```

Usage Notes

1. Program execution continues. When the ready message is displayed, normal terminal output resumes. Use the RT command to restore typing or displaying.

2. CMS error messages having a suffix letter of W, E, S, or T cannot be suppressed.

HX

Use the HX command to stop the execution of any CMS or CMS/DOS command or program, close any open files or I/O devices, and return to the CMS command environment. The format of the HX command is:

```
| HX |                                                               |
```

Usage Notes

1. HX clears all file definitions made via the FILEDEF or DLBL commands, including those entered with the PERM option.

2. The HX command is executed when the next SVC or I/O interruption occurs: therefore a delay may occur between keying HX and the return to CMS. All terminal output generated before HX is processed is displayed before the command is executed.

RO

Use the RO command, during the execution of a command or one of your programs, to resume the recording of trace information that was temporarily suspended by the SO command. Program execution continues to its normal completion, and all recorded trace information is spooled to the printer. The format of the RO command is:

```
| RO |                                                               |
```

RT

Use the RT command to restore terminal output from an executing CMS command or one of your programs that was previously suppressed by the HT command. The format of the RT command is:

RT	

Usage Note

Program execution continues, and displaying continues from the current point of execution in the program. Any terminal output that is generated after the HT command is issued and up to the time the RT command is issued is lost. Execution continues to normal program completion.

SO

Use the SO command during the execution of a command or one of your programs to temporarily suspend the recording of trace information. Program execution continues to its normal completion and all recorded trace information is spooled to the printer. The format of the SO command is:

SO	

Usage Note

To resume tracing, issue the RO command.

Section 3. EDIT Subcommands and Macros

This section describes the formats and operands of the EDIT subcommands and macros. EDIT subcommands are valid only in the environment of the CMS editor, which is invoked with the EDIT command. The EDIT command format is described in "Section 2. CMS Commands."

The editor has two modes of operation: edit mode and input mode. Whenever the EDIT command is issued, edit mode is entered; when the INPUT or REPLACE subcommands are issued with no operands, input mode is entered. In input mode, all lines you enter are written into the file you are editing. To return to edit mode from input mode, you must enter a null line (one that has no data on it).

For a functional description of the CMS editor and tutorial information on how to use it, consult the _VM/370 CMS User's Guide_.

For a summary of the default settings assumed by the editor for CMS reserved filetypes, see "Appendix A: Reserved Filetype Defaults."

EDIT Subcommands

The EDIT subcommands are listed in alphabetical order for easy reference. Each subcommand description includes the format, a list of operands (if any), usage notes, and responses. For those subcommands that operate somewhat differently on a 3270 display terminal than on a typewriter terminal, an additional discussion, "Display Mode Considerations, " is added.

Subcommands that are valid only with 3270 display terminals, namely SCROLL, SCROLLUP, and FORMAT have the notation "(3270 only)" next to the subcommand names. The FORWARD and BACKWARD subcommands, which were designed for use with 3270 terminals but can be issued at any terminal, have the notation "(primarily 3270)" next to the subcommand names.

ALTER

Use the ALTER subcommand to change a specific character to another character, one that may not be available on your terminal keyboard. The ALTER subcommand allows you to reference characters by their hexadecimal values. The format of the ALTER subcommand is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│                │                   ┌    ┌ ┐┐                              │
│ ALter          │   char1 char2     │n   │G││                              │
│                │                   │*   │*││                              │
│                │                   │1   │ ││                              │
│                │                   └    └ ┘┘                              │
└──────────────────────────────────────────────────────────────────────────┘
```

where:

char1 specifies the character to be altered. It may be specified either as a single character or as a pair of hexadecimal digits (00 through FF).

char2 specifies the character to which char1 is to be altered. It may be specified either as a single character or as a pair of hexadecimal digits.

n indicates the number of lines to be searched for the specified character. If you specify an asterisk (*), all lines in the file, beginning with the current line, are searched. If this option is omitted, then cnly the current line is searched.

G requests the editor to alter every occurrence of char1 in the lines specified. If G or * is not specified, only the first occurrence of char1 in each line specified is altered.

Usage Notes

1. If char2 is a hexadecimal value that cannot be represented on your terminal, it may appear as a blank, for example:

   ```
   input XSLC
   alter X 02
    SLC
   ```

   Column 1 contains an X'02', which cannot be displayed.

2. Use the ZONE subcommand if you want only particular columns searched for a specific character.

Responses

When verification is on, altered lines are displayed at your terminal.

Display Mode Considerations

When you request a global change on a 3270, the display is changed only once, to reflect the final position of the current line pointer. The editor displays a message to indicate the number of lines changed:

```
{ nnnn } LINE(S) CHANGED
{ NO   }
```

AUTOSAVE

Use the AUTOSAVE subcommand to set, reset, or display the automatic save function of the editor. When the automatic save function is in effect, the editor automatically issues the SAVE subcommand each time the specified number of changes or insertions are made. The format of the AUTOSAVE subcommand is:

```
┌─────────────────────────────────────────────────────────────────────┐
│              │ ┌   ┐                                                │
│ AUTOsave     │ │n  │                                                │
│              │ │OFF│                                                │
│              │ └   ┘                                                │
└─────────────────────────────────────────────────────────────────────┘
```

where:

n is a decimal number between 1 and 32767, indicating the frequency of the automatic save function. One SAVE subcommand is issued for every n lines that are changed, deleted, or added to the file.

OFF turns off the automatic save function. This is the initial setting.

Usage Notes

1. Each line affected by the $MOVE macro is treated as one update. However, all changes caused by a single CHANGE, DELETE, DSTRING, GETFILE, or OVERLAY subcommand are treated as a single update, no matter how many lines are affected.

2. If you are editing a file on a read-only disk, and an automatic save request occurs, the message:

   ```
   SET NEW FILEMODE AND RETRY
   ```

   is issued. You can enter CMS subset and access the disk in read/write mode, or use the FMODE subcommand to change the filemode to the mode of a read/write disk. If you were in input mode, you are placed in edit mode.

3. The message "SAVED" is displayed at the terminal each time the save operation occurs.

Responses

If you issue the AUTOSAVE subcommand with nc operands, the editor displays the current setting of the automatic save function.

BACKWARD (Primarily 3270)

Use the BACKWARD subcommand to move the current line pointer towards the beginning of the file you are editing. The format of the BACKWARD subcommand is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│             │ ┌ ┐                                                        │
│  BAckward   │ │n│                                                        │
│             │ │1│                                                        │
│             │ └ ┘                                                        │
└──────────────────────────────────────────────────────────────────────────┘
```

<u>where</u>:

n is the number of records backward you wish to move the current line pointer. If n is not specified, the current line pointer is moved backward one line, toward the top of the file.

Usage Note

The BACKWARD subcommand is equivalent to the UP subcommand; it is provided for the convenience of 3270 users.

Responses

When verification is on, the current line on the screen contains the record located by the BACKWARD n value. on the screen contains the record located by the BACKWARD n value. If n exceeds the number of records above the current line, TOF is displayed on the current line.

On a typewriter terminal the new current line is typed if verification is on.

BOTTOM

Use the BOTTOM subcommand to make the last line of the file the new current line. The format of the BOTTOM subcommand is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│ Bottom      │                                                            │
└──────────────────────────────────────────────────────────────────────────┘
```

Usage Note

Use the BOTTOM subcommand followed by the INPUT subcommand to begin entering new lines at the end of a file.

Responses

When verification is on, the last line in the file is displayed.

Display Mode Considerations

If the BOTTOM subcommand is issued at a 3270 display terminal in display mode, EOF: is displayed on the line following the current line, preceded by the last records of the file; the rest of the screen's output area is blank.

CASE

Use the CASE subcommand to indicate how the editor is to process uppercase and lowercase letters. The format of the CASE subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
│              │ ┌   ┐                                                 │
│ CASE         │ │M  │                                                 │
│              │ │U  │                                                 │
│              │ └   ┘                                                 │
└──────────────────────────────────────────────────────────────────────┘
```

where:

M indicates that the editor is to accept any mixture of uppercase and lowercase letters for the file as they are entered at the terminal.

U indicates that the editor is to translate all lowercase letters to uppercase letters before the letters are entered into the file. U is the default value for all filetypes except MEMO and SCRIPT.

Responses

If you enter the CASE subcommand with no operand, the current setting is displayed at the terminal.

Display Mode Considerations

If you specify CASE M when using a 3270 that does not have the lowercase feature (RPQ), you can key in lowercase characters, but they appear cn the screen as uppercase characters.

CHANGE

Use the CHANGE subcommand to change a specified group of characters to another group of characters of the same or a different length. You may use the CHANGE subcommand to change more than one line at a time. The format of the CHANGE subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
│              │                     ┌  ┌ ┐┐                           │
│ Change       │ [/string1[/string2[/│n │G││]]]                        │
│              │                     │* │*││                           │
│              │                     │1 │ ││                           │
│              │                     └  └ ┘┘                           │
└──────────────────────────────────────────────────────────────────────┘
```

where:

/ (diagonal) signifies any unique delimiting character that does not appear in the character strings involved in the change.

string1 specifies a group of characters to be changed (old data). string1 may be a null string.

string2 specifies the group of characters that are to replace string1 (new data). string2 may be a null string; if omitted, it is assumed null.

n or * indicates the number of lines to be searched, starting at the current line. If * is entered, the search is performed until the end of the file is reached. If this option is omitted, then only one line is searched.

G or * requests the editor to change every occurrence of string1 in the lines specified. If G or * is not specified, only the first occurrence of string1 in each line specified is changed. If string1 is null, G or * may not be specified.

<u>Usage Notes</u>

1. The first nonblank character following the CHANGE subcommand (or any of its truncations) is considered the delimiter. For example:

   ```
   c.VM/370.CMS.*
   ```

   changes the first occurrence of VM/370 to CMS on every line from the current line to the end of the file.

2. If string2 is omitted, it is assumed to be a null string. For example:

   ```
   THIS ISN THE LINE.
   change /n
   THIS IS THE LINE.
   ```

   A null string causes a character deletion. If string1 is null, characters are inserted at the beginning of the line. For example:

   ```
   THIS IS THE LINE.
   change //SO /
   SO THIS IS THE LINE.
   ```

3. To change multiple occurrences of the same string on one line, enter:

   ```
   change/string1/string2/ 1 *
   ```

4. The CHANGE subcommand can be used on typewriter terminals to display, without changing, any lines that contain the information specified in string1. Enter:

   ```
   change /string1/string1/ * *
   ```

5. Use the ZONE subcommand to indicate which columns are to be searched for string1. If string1 is wider than the current zone, you receive the message:

   ```
   ZONE ERROR
   ```

   and you should either reenter the CHANGE subcommand or change the zone setting.

6. If the character string inserted causes the data line to extend beyond the truncation column or the zone column, any excess characters are truncated. (See the description of the TRUNC subcommand for additional information on truncation.)

7. You should use the ALTER subcommand when you want to change a single character to some special character (one that is not available on your keyboard).

8. When the IMAGE subcommand is set with the CANON operand, backspace characters at the beginning or end of string1 are ignored.

9. To stack a CHANGE subcommand with no operands from a fixed-length EXEC, you should use the &STACK control statement.

Responses

When verification is on, every line that is changed is displayed.

Display Mode Considerations

If you issue the CHANGE subcommand without operands at a 3270 display terminal in display mode, the following occurs:

1. The record pointed to by the current line pointer appears in the user input area of the display. If the line is longer than the current truncation setting, it is truncated.

2. You can then alter the record in the user input area by retyping part or all of the line, or by using the Insert, Delete, or Erase EOF keys to insert or delete characters.

3. When the line is modified, press the Enter key, which causes the record in the user input area to replace the old record at the current line in the output display area.

If you bring a line down to the user input area and decide not to change it, press the Erase Input key and then the Enter key, and the line is not changed.

When a line is moved to the user input area, all nonprintable characters (including tabs, backspaces, control characters, and so on) are stripped from the line. Also, any characters currently assigned to VM/370 logical line editing symbols (#, @, ¢, ") are reinterpreted when the line is reentered. You should issue an explicit CHANGE subcommand to change lines containing special characters.

The CHANGE subcommand is treated as an invalid subcommand if it is issued without operands at a typewriter terminal or at a 3270 display terminal that is not in display mode.

When you request a global change on a 3270 terminal, the display is changed only once, to reflect the final position of the current line pointer. The editor displays, in the message area of the display screen:

$$\left\{\begin{array}{l}\text{nnnn}\\ \text{NO}\end{array}\right\}\ \text{LINE(S) CHANGED}$$

to indicate the number of lines that were updated. If the change request resulted in the truncation of any lines, the message is displayed as:

```
nnnn LINE(S) CHANGED nnnn LINE(S) TRUNCATED
```

If the change request moves the current line pointer beyond the end of the file, the word EOF: is displayed on the current line, preceded by the last records of the file. The rest of the output area is blank.

CMS

Use the CMS subcommand to cause the editor to enter the CMS subset mode, where you may execute those CMS commands that do not need to use the main storage being used by the editor. The format of the CMS subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
| CMS          |                                                       |
└──────────────────────────────────────────────────────────────────────┘
```

Usage Notes

1. In CMS subset, you can execute any CMS command that is nucleus-resident or that executes in the transient area. The nucleus-resident CMS commands are:

   CP
   DEBUG
   ERASE
   FETCH
   GENMOD
   INCLUDE
   LOAD
   LOADMOD
   START
   STATE
   STATEW

   The commands that execute in the transient area are:

   ACCESS
   ASSGN
   COMPARE
   DISK
   DLBL
   FILEDEF
   GENDIRT
   GLOBAL
   HELP
   LISTFILE
   MODMAP
   OPTION
   PRINT
   PUNCH
   QUERY
   READCARD
   RELEASE
   RENAME
   SET
   SVCTRACE
   SYNONYM
   TAPE
   TYPE

   To return to edit mode, use the CMS subset command RETURN.

2. If you attempt to execute a CMS command that requires main storage, you receive the message:

   INVALID SUBSET COMMAND

   Results are unpredictable at this point. You should not attempt to execute any program that executes in the user program area. Using the LOAD, INCLUDE (RESET), FETCH, START, and RUN commands could load programs that would overlay the editor's storage area and its contents. Use these commands only for programs that execute in the transient area.

3. In an edit macro, if you attempt to use a command that is invalid in the CMS subset, you receive a return code of -0002.

4. If you attempt to execute a CMS command that fails because of insufficient storage, your EDIT session may abnormally terminate. You should save input you have entered before you enter CMS subset mode.

Responses

After you issue the CMS subcommand, you receive the message:

```
     CMS SUBSET
```

to indicate that you are in CMS subset mode. On a display terminal, the screen is cleared before the editor issues this message; the display of the file is restored when you enter the RETURN command.

DELETE

Use the DELETE subcommand to delete one or more lines from a file, beginning with the current line. The line immediately following the last line deleted becomes the new current line. The format of the DELETE subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
|             |  ┌ ┐                                                    |
| DELete      |  |n|                                                    |
|             |  |*|                                                    |
|             |  |1|                                                    |
|             |  └ ┘                                                    |
└──────────────────────────────────────────────────────────────────────┘
```

where:

n   indicates the number of lines to be deleted, starting at the current line. If an asterisk (*) is entered, the remainder of the file is deleted. If n is omitted, only one line is deleted.

Responses

None. If you delete the last line in the file, or if you issue the DELETE subcommand when the current line pointer is already at the end of the file, the editor displays the message:

```
EOF:
```

Display Mode Considerations

If you delete a record when using a display terminal in display mode, the editor rewrites the output display area with the records above the current line pointer unchanged. The record at the current line pointer and the remaining records on the screen move up by one, and a new record (if one exists) moves into the bottom of the output display area.

DOWN

Use the DOWN subcommand to advance the current line pointer forward in the file. The line pointed to becomes the new current line. The format of the DOWN subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
|             |  ┌ ┐                                                    |
| DOwn        |  |n|                                                    |
|             |  |1|                                                    |
|             |  └ ┘                                                    |
└──────────────────────────────────────────────────────────────────────┘
```

where:

n   indicates the number of lines to advance the pointer, starting at the current line. If n is not specified, the current line pointer is advanced one line.

Usage Note

DOWN is equivalent to the NEXT and FORWARD subcommands.

Responses

When verification is on, the new current line is displayed at the terminal; if the end of the file is reached, the message:

```
    EOF:
```

is displayed.

DSTRING

Use the DSTRING subcommand to delete one or more lines beginning with the current line, down to, but not including, the first line containing a specified character string. The current line is not checked for the character string. The format of the DSTRING subcommand is:

```
| DString    | /[string[/]]
```

where:

/ (diagonal) signifies any unique delimiting character that does nct appear in the string.

string specifies the group of characters for which a search is to be made. If string is not specified, only the current line is deleted.

Usage Note

The zone set by the ZONE subcommand or the default zone setting is checked for the presence of the character string. A character string with a length greater than the current zone setting causes the error message ZONE ERROR.

Responses

If the character string is not found by the end of the file, no deletions occur, the current line pointer is unchanged, and the message:

```
     STRING NOT FOUND, NO DELETIONS MADE
```

is displayed.

Display Mode Considerations

If verificaticn is on when the DSTRING subcommand is issued at a display terminal in display mode, the screen is changed to reflect the deletions from the file.

FILE

Use the FILE subcommand to write the edited file on disk and, optionally, override the file identifier originally supplied in the EDIT command. The format of the FILE subcommand is:

FILE	[fn [ft [fm]]]

where:

fn indicates the filename for the file. If filename is omitted, filetype and filemode cannot be specified, and the existing filename, filetype, and filemode are used.

ft indicates the filetype for the file.

fm indicates the filemode for the file.

Usage Notes

1. When you specify a file identifier, any existing file that has an identical fileid is replaced. If the file being edited had been previously written to disk, that copy of the file is not altered.

2. You can change the filename and filemode during the editing session using the FNAME and FMODE subcommands.

Responses

The CMS ready message indicates that the file has been written to disk and control is returned to the CMS environment.

FIND

Use the FIND subcommand to locate a line based on its initial character string. The format of the FIND subcommand is:

Find	[line]

where:

line is any character string, including blanks and tabs, that you expect to find beginning in column 1 of an input record. At least one non-blank character must be specified. If line is not specified, the current line pointer is moved down one line.

Usage Notes

1. Only one blank can be used as a delimiter following the FIND subcommand; additional blanks are considered part of the character string.

2. If the image setting is ON, the editor expands tab characters to the apprcpriate number of blanks before searching for the line.

3. If the current line pointer is at the bottom of the file when the FIND subcommand is issued the search begins at the top of the file.

Responses

When verification is on, the line is displayed at the terminal. If the line is not found, the message:

```
EOF:
```

is displayed and you may use the REUSE (=) subcommand to search again, beginning at the top of the file.

FMODE

Use the FMODE subcommand to display or change the filemode of a file. The format of the FMODE subcommand is:

FMode	[fm]

where:

fm indicates the filemode that is to replace the current filemode setting. You can specify a filemode letter (A-Z) or a filemode letter and number (0-5). If you specify a filemode letter, the existing filemode number is retained.

Usage Notes

1. The specified filemode is used the next time a FILE, SAVE, or automatic save request is issued. If the file being edited had been previously filed or saved, that copy of the file remains unchanged.
2. If the disk specified by filemode already contains a file with the same filename and filetype, that file is replaced when a FILE, SAVE, or automatic save request is issued; no warning message is issued.
3. If the filemode specified is that of a read-only disk, then when an attempt is made to file or save the file, the editor displays an error message.

Responses

If you enter the FMODE subcommand without specifying fm, the editor displays the current filemode.

Display Mode Considerations

When you specify a new filemode with the FMODE subcommand, the editor writes the new filemode in the filemode field at the top of the screen.

FNAME

Use the FNAME subcommand to display or change the filename of a file. The format of the FNAME subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
| FName       | [fn]                                                   |
└──────────────────────────────────────────────────────────────────────┘
```

where:

fn indicates the filename that is to replace the current filename.

Usage Notes

1. The specified filename is used the next time a FILE, SAVE, or automatic save request is issued. If the file being edited had been previously filed or saved, that copy of the file remains unchanged.

2. If a file already exists with the specified filename and the same filetype and filemode, that file is replaced; no warning message is issued.

3. You can use the FNAME subcommand when you want to make multiple copies of a file, with different filenames, without terminating your edit session.

Responses

If you enter the FNAME subcommand without specifying fn, the editor displays the current filename.

Display Mode Considerations

When you issue the FNAME subcommand specifying a new filename, the editor writes the new name in the filename field at the top of the screen.

FORMAT (3270 Only)

Use the FORMAT subcommand to change the mode of a local or remote 3270 terminal from display to line or line to display mode. The format of the FORMAT subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
| FORMat      | {DISPLAY}                                              |
|             | {LINE   }                                              |
└──────────────────────────────────────────────────────────────────────┘
```

where:

DISPLAY specifies that a full screen display of data is to occur. Subcommands do not appear as part of the data displayed.

LINE specifies that the display station is to operate as a typewriter terminal. Every line you enter is displayed on the screen; the screen looks like a typewriter terminal's console sheet.

Usage Notes

1. Line mode is the default for remote 3270s. If you are using a remote 3270 in display mode, and you enter the INPUT subcommand, you are placed in line mode while you enter input. When you return to edit mode, the full screen display is restored.

2. The FORMAT subcommand is treated as invalid under any of the following conditions:

   a. The NODISP option of the EDIT command was used to invoke the editor.

   b. The edit session was initiated on a typewriter terminal. (The session may optionally be continued on a 3270 after a reconnection.)

   To obtain a full screen display, you must save your file and restart your edit session.

3. The column settings for the VERIFY, TRUNC, and ZONE subcommands remain unchanged when you issue the FORMAT subcommand.

Responses

None.

FORWARD (Primarily 3270)

Use the FORWARD subcommand to move the current line pointer towards the end of the file you are editing. The format of the FORWARD subcommand is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│              │ ┌ ┐                                                        │
│  FOrward     │ │n│                                                        │
│              │ │1│                                                        │
│              │ └ ┘                                                        │
└──────────────────────────────────────────────────────────────────────────┘
```

where:

n is the number of records you wish to move forward in the file being edited. If n is not specified, 1 is assumed.

Usage Note

The FORWARD subcommand is equivalent to the DOWN and NEXT subcommands; it is provided for the convenience of 3270 users.

Responses

When verification is on, the new current line is displayed. If the number specified exceeds the number of lines remaining in the file, the current line pointer is positioned at EOF:.

GETFILE

Use the GETFILE subcommand to insert all or part of a specific CMS file into a file that you are editing. The format of the GETFILE subcommand is:

```
┌─────────────────────────────────────────────────────────────────────────────┐
|               |      ┌    ┌    ┌           ┌        ┐┐┐┐                   |
| Getfile       | fn   |ft  |fm  |firstrec   |numrec||||                   |
|               |      |    |    |1          |*       ||||                   |
|               |      └    └    └           └        ┘┘┘┘                   |
└─────────────────────────────────────────────────────────────────────────────┘
```

where:

fn is the filename of the file that contains the data to be inserted into the file you are editing.

ft is the filetype of the file that contains the data to be inserted. If ft is not specified, the filetype of the file you are editing is assumed.

fm is the filemode of the file that contains the data to be inserted. If fm is not specified, all of your accessed disks are searched for the file.

firstrec indicates the record number of the first record you want to copy.

numrec indicates the number of lines to be inserted, starting with the line specified by firstrec. If numrec is not specified, or specified as *, then the remainder of the file between firstrec and the end of the file is inserted.

Usage Notes

1. The GETFILE operand list is positional; if you omit one operand, you cannot specify any operands that follow. Thus, if you want to specify firstrec and lastrec, you must specify the filetype and filemode of the file.

2. The last line inserted becomes the new current line.

3. If the record length of the records in the file containing the data to be inserted exceeds that of the file being edited, an errcr message is displayed, and the GETFILE is not executed; if shorter, the reccrds are padded to the record length of the file being edited and inserted in the file.

4. If you use the GETFILE subcommand to insert lines into a VSBASIC file, you must also use the RENUM subcommand to resequence the file.

5. If the editor fills up available storage while executing a GETFILE request, it may not be able to copy all of the file. You should determine how many records were actually copied, and then write the current file on disk.

Responses

When verification is on, the last line inserted into the file is displayed. If the end of the file has been reached, the message:

```
EOF REACHED
```

is displayed, followed by the display of the last line inserted.

IMAGE

Use the IMAGE subcommand to control how the editor should handle backspaces and tab characters or to display the current image setting. The format of the IMAGE subcommand is:

```
┌──────────────────────────────────────────────────────────────────┐
│            │ ┌      ┐                                             │
│ IMAGE      │ │ON    │                                             │
│            │ │OFF   │                                             │
│            │ │CANON │                                             │
│            │ └      ┘                                             │
└──────────────────────────────────────────────────────────────────┘
```

where:

ON specifies that any text entered while in input mode or as a line of data following a FIND, INPUT, OVERLAY, or REPLACE subcommand, is expanded into a line image; backspaces are removed and tabs are replaced by blanks.

Text entered in the form of delimited strings, as in CHANGE, LOCATE, and ALTER, is not expanded; tabs and backspaces are treated in the same way as other characters.

IMAGE ON is the default for all filetypes except SCRIPT.

OFF specifies that tabs and backspaces are treated as data characters in the same way as other characters. They are not deleted, translated, expanded, or reordered.

CANON specifies that backspaces may be used to produce compound characters such as underscored words, headings, or phrases. Before they are inserted in the file, compound characters are ordered, with backspaces arranged singly between the characters that overlay each other; the overlaying characters are arranged according to their EBCDIC values. Tab characters are handled as for IMAGE OFF.

CANON is the default for SCRIPT files.

Usage Notes

1. When the image setting is ON, tab characters are expanded to an appropriate number of blanks, according to the current settings of the TABSET subcommand. The TABSET command has no effect if the image setting is either OFF or CANON.

2. When the image setting is on, backspaces are handled as follows:

   - Backspace characters act in a similar manner to the logical character delete symbol, in deleting the previous characters if a sufficient number of other characters or blanks follow the backspace characters. However, backspace characters that immediately follow a command name are interpreted as separator characters and do not delete any part of the command name.

   - If a backspace character is the last character in the input line, it is ignored.

Responses

When you issue the IMAGE subcommand with no operand, the current IMAGE setting is displayed.

INPUT

Use the INPUT subcommand to insert a single line into a file, or, if no data line is specified, to leave edit mode and enter input mode. The format of the INPUT subcommand is:

```
| Input      | [line]                                               |
```

where:

line specifies the input line to be entered into the file. It can contain blanks and tabs; if you enter at least two blanks following the INPUT subcommand and no additional text, a blank line is inserted into the file.

Usage Notes

1. Each line that is inserted into the file becomes the new current line.

2. When you are using line-number editing (LINEMODE LEFT or LINEMODE RIGHT) you cannot use the INPUT subcommand to insert a single line of data; use the nnnnn subcommand.

3. To stack an INPUT subcommand in order to enter input mode from a fixed-length EXEC, you should use the &STACK control statement.

Responses

When you issue the INPUT subcommand without operands, and verification is on, the editor displays:

```
INPUT:
```

All subsequent lines you entered are written into the file, until you enter a null line to return to edit mode.

Display Mode Considerations

1. When you insert lines while using a local display terminal in display mode, the editor writes each record on the current line. The old current line and all records above it move up one line, except for the topmost record formerly on line 2, which is deleted from the screen.

2. If you are using a remote display terminal in display mode and you issue the INPUT subcommand with no text, the terminal is forced into line mode. The display of the file on the screen disappears and the word INPUT: appears. As you enter input lines, they appear in the output display area. When you leave input mode by entering a null line, the remote terminal returns to display mode. The display of the file reappears on the screen, with the lines you have just entered in their proper place in the file.

3. When you are entering data in input mode at a display terminal that is in line mode, a tab character generated by a program function (PF) key only generates one character, and appears as one character on the screen. That is, the line does not appear spaced according to the tab settings.

LINEMODE

Use the LINEMODE subcommand to set, cancel, or display the status of line-number editing. When you use line-number editing, you can input, locate, and replace lines by referencing their record numbers. Line-number editing is the default for VSBASIC and FREEFORT files. The format of the LINEMODE subcommand is:

```
┌────────────────────────────────────────────────────────────────────┐
|                 | ┌      ┐                                         |
| LINEmode        | |LEFT  |                                         |
|                 | |RIGHT |                                         |
|                 | |OFF   |                                         |
|                 | └      ┘                                         |
└────────────────────────────────────────────────────────────────────┘
```

where:

LEFT
L
initializes line-number editing and places sequence numbers on the left, in columns 1 through 5, right-justified and padded with blanks; the near zone is set to 7. If the filetype is FREEFORT, columns 1 through 8 are used for serial numbers; the near zone is set to 9.

You should never use left-handed line-number editing for files in which data must occupy columns 1 through 6, for example ASSEMBLE files.

RIGHT
R
initializes line-number editing and places sequence numbers on the right, in columns 76 to 80, right-justified and padded with zeroes. The end zone and truncation columns are set to 72.

This operand is valid only for files with fixed-length 80-character records.

OFF cancels line-number editing and (if you were using left-handed line-number editing) resets the first logical tab setting to column 1. The VERIFY, TRUNC, and ZONE subcommand settings remain unchanged. Serialization may still be in effect. OFF is the default for all filetypes except VSBASIC and FREEFORT.

<u>Note</u>: If you enter LINEMODE OFF while editing a FREEFORT file, line-number editing cannot be resumed for the remainder of the edit session.

<u>Usage Notes</u>

1. When you enter input mode while you are using line-number editing, you are prompted with a line number to enter each line. The default prompting increment is 10; you may change it using the PROMPT subcommand.

   If you enter input mode after using the nnnnn subcommand to position the current line pointer, the prompted line number is the next higher multiple of the current prompting increment or an adjusted line number, whichever is smaller. The adjusted line number is determined according to the following formula:

   $$\text{pppp} = 1 + \text{cccc} + \frac{\text{nnnn - cccc}}{4}$$ (Any fractional remainder is dropped.)

   where:

   pppp is the prompt line number.

   cccc is the current line number.

   nnnn is the next sequential line number in the file.

2. When you are prompted on a typewriter terminal, enter your input line on the same line as the prompted line number. If you are using right-handed line-number editing, on a typewriter terminal or on a display terminal in line mode, the serial numbers are not redisplayed in columns 76 to 80 (unless you use the VERIFY subcommand to increase the verification setting). When a line is displayed in edit mode, the line numbers always appear on the left even though they are on the right in the disk copy of the file. Whether or not the line numbers are displayed on the right depends on the current verification setting.

3. You cannot use the INPUT or REPLACE subcommands to input a single data line when you are using line-number editing; use the nnnnn subcommand instead.

4. When you initialize line-number editing for files that already exist, the editor assumes that the records are in the proper format and numbered in ascending order.

5. If you want to place serial numbers in columns 76 through 80, but you do not wish to use line-number editing, use the SERIAL subcommand.

<u>Responses</u>

When you issue the LINEMODE subcommand with no operands, the current setting is displayed.

Display Mode Considerations

When you use line-number editing on a display terminal in display mode, the prompting numbers in input mode appear on line 2 of the display screen, in the editor message area. Enter your input lines in the user input area. Regardless of whether you are using right- or left-handed line-number editing, the line numbers always appear in their true position in the file.

LOCATE

Use the LOCATE subcommand to scan the file beginning with the next line for the first occurrence of a specified character string. The format of the LOCATE subcommand is:

```
|[Locate]    | /[string[/]]                                    |
```

where:

/ (diagonal)   signifies any unique delimiting character that does not appear in the string. The delimiter may be any nonblank character. The closing delimiter is optional.

string   specifies any group of characters to be searched for in the file.

Usage Notes

1. If the beginning delimiter is /, you can omit the subcommand name LOCATE. If you enter only:

   /

   on a line, the current line pointer is moved down one line.

2. If string is null or blank, the search is successful on the first line encountered. If the line pointer is at the end of the file when the LOCATE subcommand is issued, scanning starts from the top of the file.

3. Use the ZONE subcommand when you want the editor to search only a specific column. If you specify a character string longer than the current zone width, the editor issues the message ZONE ERROR.

Responses

When verification is on, the line containing the specified string is displayed. If the string is not found, the messages:

```
NOT FOUND
EOF:
```

are displayed, and you may use the REUSE (=) subcommand to request that command be repeated, beginning at the top of the file.

LONG

Use the LONG subcommand to cancel a previous SHORT subcommand request. The format of the LONG subcommand is:

LONG	

Usage Note

When the LONG subcommand is in effect (it is the default), the editor responds to invalid subcommands with the message:

```
?EDIT: line ...
```

Responses

None.

NEXT

Use the NEXT subcommand to advance the line pointer a specified number of lines toward the end of the file. The line pointed to becomes the new current line. The format of the NEXT subcommand is:

Next	n 1

where:

n indicates the number of lines to move the line pointer. If n is omitted, then the pointer is moved down only one line.

Usage Note

NEXT is equivalent to DOWN and FORWARD.

Responses

When verification is on, the new current line is displayed. If the end of the file is reached, the message:

```
EOF:
```

is displayed.

OVERLAY

Use the OVERLAY subcommand to selectively replace one or more character strings in the current line with the corresponding nonblank characters in the line being keyed in. The format of the OVERLAY subcommand is:

```
| Overlay    | [line]                                                  |
```

where:

line specifies an input line that replaces corresponding character positions in the current line. On a typewriter terminal, if you enter the OVERLAY subcommand with no data line, the input record remains unchanged.

Usage Notes

1. Blank characters in the input line indicate that the corresponding characters in the current line are not to be overlaid. For example:

   ```
   CHARMIE
   o     L
   CHARLIE
   ```

   Blanks in columns 3, 4, 5, and 6 of the OVERLAY line indicate that columns 1, 2, 3, and 4 of the current line are not to be changed. (At least one blank must follow the OVERLAY subcommand, which can be truncated as O).

2. This subcommand may be entered at a typewriter terminal by typing the letter "o", followed by a backspace, followed by the overlaying characters. This sets up the correct alignment on the terminal.

3. An underscore in the overlaying line must be used to place a blank into the corresponding position of the current line. Thus, an underscore cannot be placed (or replaced) in a line.

   OVERLAY should be used with care on lines containing underscored words or other compound characters.

4. To perform a global overlay operation, issue the REPEAT subcommand just prior to issuing the OVERLAY subcommand. For example, when you enter:

   ```
   repeat *
   overlay X
   ```

   an X is placed in the leftmost column of each record in the file, beginning with the current line. The leftmost column, for files with the IMAGE setting ON, is determined by the first logical tab setting.

Responses

When verification is on, the line is displayed at the terminal after it has been overlaid.

Display Mode Considerations

In addition to using the OVERLAY subcommand in the normal way, you may also issue the OVERLAY subcommand with no operands. The next line you enter is treated as overlay data. To cancel the overlay request, press the Erase Input key and then the Enter key.

PRESERVE

Use the PRESERVE subcommand to save the settings of various EDIT subcommands until a subsequent RESTORE subcommand is issued. The format of the PRESERVE subcommand is:

```
┌──────────────────────────────────────────────────────────────────┐
│ PREserve   │                                                     │
└──────────────────────────────────────────────────────────────────┘
```

Usage Note

Settings are saved for the following subcommands:

CASE
FMODE
FNAME
IMAGE
LINEMODE
LONG
PROMPT
RECFM
SERIAL
SHORT
TABSET
TRUNC
VERIFY
ZONE

Responses

None.

PROMPT

Use the PROMPT subcommand to change the prompting increment for input line numbers when you are using line-number editing. The format of the PROMPT subcommand is:

```
┌──────────────────────────────────────────────────────────────────┐
│              │ ┌  ┐                                              │
│ PROMPT       │ │n │                                              │
│              │ │10│                                              │
│              │ └  ┘                                              │
└──────────────────────────────────────────────────────────────────┘
```

where:

n   specifies the prompting increment; the default value is 10. The value cf n should not exceed 32,767.

Responses

When you issue the PROMPT subcommand with no operands, the current setting is displayed.

QUIT

Use the QUIT subcommand to terminate the current editing session and leave the previous copy of the file, if any, intact on the disk. The format of the QUIT subcommand is:

```
┌────────────────────────────────────────────────────────────┐
│ QUIT        │                                              │
└────────────────────────────────────────────────────────────┘
```

Usage Notes

1. You can use the QUIT subcommand when you have made a global change that introduced errors into your file; or whenever you discover that you have made errors in editing a file and want to cancel your editing session.

   If a SAVE subcommand or automatic save request has been issued, the file remains as it was when last written.

2. The QUIT subcommand is a convenient way to terminate an edit session when you enter an incorrect filename on the EDIT command line, or when you edit a file merely to examine, but not to change, its contents.

Responses

The CMS ready message indicates that control has been returned to CMS.

RECFM

Use the RECFM subcommand to indicate to the editor whether the record format of the file is fixed-length or variable-length, or to display the current RECFM setting. The format of the RECFM subcommand is:

```
┌────────────────────────────────────────────────────────────┐
│             │                                              │
│ RECfm       │ ┌ ┐                                          │
│             │ │F│                                          │
│             │ │V│                                          │
│             │ └ ┘                                          │
└────────────────────────────────────────────────────────────┘
```

where:

F   indicates fixed-length records.

V   indicates variable-length records.

Usage Notes

1. V is assumed by default for all new EXEC, LISTING, FREEFORT, VSBDATA, and SCRIPT files. Usually, a variable-length format file occupies a smaller amount of disk space because trailing blanks are deleted from each line before it is written onto disk. When variable-length VSBDATA files are written to disk, however, trailing blanks are not truncated (to allow VSBDATA file to span records).

2. When you use the RECFM subcommand to change the format of a file from fixed-length to variable-length records, trailing blanks are removed when the file is written to disk; when you are changing variable-length records to fixed-length, all records are padded to the record length.

Responses

When you use the RECFM subcommand without specifying F or V, the current setting is displayed.

Display Mode Considerations

When you specify a new record format with the RECFM subcommand, the editor writes the new record format in the format field at the top of the screen.

RENUM

Use the RENUM subcommand to recompute the line numbers for VSBASIC and FREEFORT source files. The format of the RENUM subcommand is:

```
┌──────────────────────────────────────────────────────────────────┐
│               │ ┌        ┌      ┐┐                               │
│ RENum         │ │strtno  │incrno││                               │
│               │ │ 10     │strtno││                               │
│               │ └        └      ┘┘                               │
└──────────────────────────────────────────────────────────────────┘
```

where:

strtno indicates the number from which you wish to start renumbering your file. Because RENUM renumbers the whole file from beginning to end, the number you specify as strtno becomes the statement number of the first statement in the newly renumbered file. This number may not exceed 99999 for VSBASIC files or 99999999 for FREEFORT files. The default start number value is 10 and the specified start number must not be zero.

incrno indicates the increment number value by which you wish to renumber your file. This value may not exceed 99999 for VSBASIC files or 99999999 for FREEFORT files. The default for incrno is strtno, the first sequence number in the renumbered file, and the specified incrno must not be zero.

Usage Notes

1. If you do not specify strtno and incrno, the default value for both is 10. If you specify only strtno, incrno defaults to the same value as strtno.

2. The current line pointer remains as it was before you entered the RENUM subcommand regardless of whether or not RENUM completes successfully. If you are editing a VSBASIC file, the file to be renumbered must either originate from a read/write disk or you must issue an FMODE subcommand to change the file destination to a read/write disk.

3. All VSBASIC statements that use statement numbers for operands are updated to reflect the new line numbers. The VSBASIC statements with line number operands are:

    CLOSE
    CLOSEFILE
    DELETE
    EXIT
    GET
    GOSUB
    GOTO
    IF
    ON
    OPEN
    OPENFILE
    PRINT USING
    PUT
    READFILE
    REREADFILE
    RESET
    RESETFILE
    REWRITEFILE
    WRITEFILE

4. If any error occurs during the RENUM operation, the editor terminates the RENUM operation and the file being edited remains unchanged.

<u>Responses</u>

When verification is on, the message EDIT: indicates that the RENUM subcommand completed processing.

REPEAT

Use the REPEAT subcommand to execute the immediately following OVERLAY subcommand (or an X or Y subcommand assigned to invoke OVERLAY) for the specified number of lines or to the end of the file. The format of the REPEAT subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
│              │ ┌ ┐                                                   │
│ REPEAT       │ │n│                                                   │
│              │ │*│                                                   │
│              │ │1│                                                   │
│              │ └ ┘                                                   │
└──────────────────────────────────────────────────────────────────────┘
```

<u>where</u>:

n   indicates the number of times to repeat the OVERLAY request that immediately follows, beginning with the current line. An asterisk (*) indicates that the request is to be repeated until the end of the file is reached. If neither n nor * is specified, then only one line is handled. The last line processed becomes the new current line.

<u>Usage Notes</u>

1. If the next subcommand issued after the REPEAT subcommand is not an OVERLAY subcommand, the REPEAT subcommand is ignored.

2. For an example of a REPEAT subcommand followed by an OVERLAY subcommand, see the discussion of the OVERLAY subcommand.

<u>Responses</u>

None.

REPLACE

Use the REPLACE subcommand to replace the current line with a specified line or to delete the current line and enter input mode. The format of the REPLACE subcommand is:

Replace	[line]

where:

line specifies an input line that is to replace the current line. If a line is specified, then the editor puts it into the file in place of the current line. If no line is specified, the editor deletes the current line and enters input mode (see Usage Note 2 for exception).

Usage Notes

1. If the LINEMODE subcommand with a LEFT or RIGHT operand is in effect, then issuing the REPLACE subcommand specifying a line is not valid. If the REPLACE subcommand is used without any operands when LINEMODE is set to LEFT or RIGHT, you are prompted for the next available line number; the first data line you enter replaces the current line number.

2. If you use the REPLACE subcommand with no operands to enter input mode, and the next line you enter is a null line, then the current line is not deleted, and you are returned to edit mode.

3. To stack a REPLACE subcommand in order to enter input mode from a fixed-length EXEC, you should use the &STACK control statement.

Responses

When verification is on and you issue the REPLACE subcommand with no data line, the message:

```
INPUT:
```

indicates that your virtual machine is in input mode.

RESTORE

Use the RESTORE subcommand to restore the settings of EDIT subcommands to their values when the PRESERVE subcommand was last issued or to their default values if a PRESERVE subcommand has not been issued. The format of the RESTORE subcommand is:

REStore	

Usage Note

The settings are restored for the following subcommands:

CASE
FMODE
FNAME
IMAGE
LINEMODE
LONG
PROMPT
RECFM
SERIAL
SHORT
TABSET
TRUNC
VERIFY
ZONE

Responses

None.

RETURN

Use the RETURN subcommand to return to edit mode from the CMS subset environment. RETURN is not an EDIT subcommand, but is listed here as a companion tc the CMS subcommand. The format of the RETURN command is:

```
| RETURN     |                                                     |
```

Responses

When verification is on, the editor responds:

```
EDIT:
```

to indicate that your virtual machine is in edit mode.

REUSE (=)

Use the REUSE subcommand (which can also be specified as =) to stack last in, first out (LIFO) the last EDIT request, except for REUSE or a question mark, and then execute the stacked subcommands. The format of the REUSE (or =) subcommand is:

```
| {REUSE}   | [subcommand]                                        |
| {=    }   |                                                     |
```

where:

subcommand specifies any valid EDIT subcommand.

Usage Notes

1. If the subcommand you enter on the REUSE subcommand line is an invalid subcommand, the editor clears the stack.

2. You can use the REUSE subcommand to repeat a subcommand request that was not satisfied the first time, for example, a LOCATE subcommand that resulted in an end-of-file condition. If you enter:

   ```
   =
   ```

   the LOCATE subcommand is stacked, then read by the editor and executed again. This time the search begins from the top of the file.

3. You can also enter more than one equal sign (=) on a single line, to stack the last issued subcommand more than once. For example:

   ```
   locate /xyz/
   XYZ IS MY FAVORITE
   = = = =
   I FIRST MET XYZ
   XYZ'S NAME IS DERIVED
   LAST SAW XYZ
   EOF:
   ```

   the LOCATE subcommand is stacked four times, and then the editor, reading from the stack, executes the four stacked subcommands.

4. You can do the following if you issue a CHANGE subcommand before positioning your current line pointer:

   ```
   c/xx/yy
   NOT FOUND
   = l/x/
   LINE XXXX
   LINE YYXX
   ```

   In this example, the CHANGE request was issued and string1 was not found. The REUSE subcommand stacks the CHANGE subcommand and stacks a LOCATE subcommand in front of it. The LOCATE subcommand is read and executed, followed by the CHANGE subcommand.

5. You can stack an INPUT or REPLACE subcommand in front of a data line you mistakenly entered in edit mode, for example:

   ```
   roses are red, violets are blue
   ?EDIT: ROSES ARE RED, VIOLETS ARE BLUE
   = input
   INPUT:
   without cms
   i would be, too.
   ```

   The = subcommand stacks the INPUT subcommand in front of the data line. Reading from the stack, the editor executes the INPUT subcommand, then reads in, as the first line of data, the line beginning with ROSES. The file contains:

   ```
   ROSES ARE RED, VIOLETS ARE BLUE
   WITHOUT CMS
   I WOULD BE, TOO.
   ```

Responses

Responses are those that are issued to the stacked subcommands.

SAVE

Use the SAVE subcommand to write the file that is currently being edited onto the disk, without returning control to CMS, and optionally to change the file identifier. The format of the SAVE subcommand is:

```
┌──────────────────────────────────────────────────────────────────┐
| SAVE        | [fn [ft [fm ]]]                                    |
└──────────────────────────────────────────────────────────────────┘
```

where:

fn indicates the filename of the file to be saved. If you specify only fn, then the filetype and filemode are the same.

ft indicates the filetype of the file to be saved.

fm indicates the filemode of the file to be saved.

Usage Notes

1. If you specify a new file identifier, any existing file with the same file identifier is replaced; no message is issued. The file being edited, if previously written to disk, is not altered.
2. To write a file on disk and terminate the editing session, use the FILE subcommand.
3. If you want to save the contents of a file at regular intervals, use the AUTOSAVE subcommand.

Responses

When verification is on, the editor displays:

```
EDIT:
```

to indicate the SAVE request completed successfully and you may continue to enter EDIT subcommands.

SCROLL/SCROLLUP (3270 Only)

Use the SCROLL and SCROLLUP subcommands to scan the contents of a file on a display screen.

SCROLL causes the editor to scan forward through the file; SCROLLUP causes the editor to scan backward through the file. The format of the SCROLL and SCROLLUP subcommands is:

```
┌──────────────────────────────────────────────────────────────────┐
|                    | ┌   ┐                                       |
| {Scroll        }   | |n  |                                       |
| {S[croll]U[p]  }   | |*  |                                       |
|                    | |1  |                                       |
|                    | └   ┘                                       |
└──────────────────────────────────────────────────────────────────┘
```

where:

n is a number from 1 to 255 that specifies the number of successive screens of data to be displayed. If an asterisk (*) is specified, the entire file, from the current line to the end or beginning of the file, is displayed. If n is not specified, 1 is the default.

Usage Notes

1. The SCROLLUP subcommand can be specified by any combination of the truncation of SCROLL and UP; the minimum truncation is SU.

2. The number of lines shifted forward or backward depends on the current verification setting. If the verification setting is 80 characters or less, then a scroll request displays a file in increments equal to the number of lines that can be displayed in the output display area of the screen. equal to the number of lines that can be displayed in the output display area of the screen. If the verification setting is more than 80 characters, then a SCROLL request displays a file in increments equal to half the number of lines that can be displayed in the output area.

   Therefore, a single SCROLL on a 3270 Model 2 display terminal equal to half the number of lines that can be displayed in the output area. Therefore, on a 3270 Model 2 display terminal, is the equivalent of DOWN 20 or DOWN 10, depending on the record length, and SCROLLUP is the equivalent of UP 20 or UP 10.

3. When you use the SCROLL or SCROLLUP subcommands to display more than one screenful, each display is held for one minute, and the screen status area indicates MORE... . To hold the screen display longer, press the Enter key.

   To halt scrolling before all the requested screenfuls are displayed, enter the HT Immediate command and press the Cancel key twice.

4. When you begin scrolling from the top of the file, the first screenful contains only the first seven lines. When you scroll to the end of the file, the last screen may duplicate lines displayed in the previous screen.

Responses

The screen display is shifted forward or backward.

SERIAL

Use the SERIAL subcommand to control the serialization of records in columns 73 through 80. The format of the SERIAL subcommand is:

```
| SERial      | (OFF          ) |
|             | (     ┌    ┐  ) |
|             | (ON   |incr|  ) |
|             | (ALL  |10  |  ) |
|             | (seq  └    ┘  ) |
```

where:

OFF indicates that neither serialization numbers nor identifiers are to be placed in columns 73-80.

ON indicates that the first three characters of the filename are to be used in columns 73-75 as an identifier.

ALL indicates that columns 73-80 are to be used for serialization numbers.

seq specifies a three-character identification to be used in columns 73-75.

incr specifies the increment for the line number in columns 76-80 (or 73-80). This number also becomes the first line number. If incr is not specified, then 10 is assumed.

Usage Notes

1. The SERIAL subcommand is valid only for files with fixed-length, 80-character records. To renumber VSBASIC or FREEFORT files, use the RENUM subcommand.

2. The serialization setting is ON, by default, for the following filetypes:

   ```
   ASSEMBLE       PLI
   COBOL          PLIOPT
   DIRECT         UPDATE
   FORTRAN        UPDTxxxx
   MACRO
   ```

3. When serialization is in effect, records in a file are resequenced each time a FILE, SAVE, or AUTOSAVE request is issued. If you are using line-number editing, you must issue the subcommand:

   ```
   linemode off
   ```

   before issuing a FILE or SAVE subcommand if you wish the records to be resequenced.

Responses

If you issue the SERIAL subcommand in a file with a zone column greater than 72, the message:

```
END ZONE SET TO 72
```

is displayed, to indicate that the zone has been changed. If the zone column is 72 or less, but the truncation column is greater than 72, the message:

```
TRUNC SET TO 72
```

is displayed.

SHORT

Use the SHORT subcommand to request the editor to respond to invalid subcommand lines with the short form of the ?EDIT message. The format of the SHORT subcommand is:

```
┌────────────────────────────────────────────────────────────────────────┐
| SHORT      |                                                           |
└────────────────────────────────────────────────────────────────────────┘
```

<u>Usage Notes</u>

1. When the SHORT subcommand is in effect, the editor responds:

   ¬

   to an invalid subcommand line, and responds:

   ¬$

   to an invalid macro request.

2. To resume displaying the long form of the ?EDIT message, use the LONG subcommand.

<u>Responses</u>

None.

STACK

Use the STACK subcommand to stack data lines or EDIT subcommands in the console stack for subsequent reading. The format of the STACK subcommand is:

```
┌────────────────────────────────────────────────────────────────────────┐
|            | ┌          ┐                                              |
|            | |n         |                                              |
| STACK      | |subcommand|                                              |
|            | |0         |                                              |
|            | |1         |                                              |
|            | └          ┘                                              |
└────────────────────────────────────────────────────────────────────────┘
```

<u>where</u>:

n — indicates the number of lines to be stacked beginning with the current line. If a number or a subcommand is nct specified, then one line is assumed by default. A maximum of 25 lines can be stacked.

If the current line pointer is at the top of the file, then n-1 lines are stacked. If fewer than n lines remain in the file, only the lines remaining are stacked.

subcommand — specifies an EDIT subcommand to be stacked.

0 — stacks a null line.

Usage Notes

1. STACK subcommands are used to write edit macros, to stack lines from a file so that they can be moved around, or to stack additional subcommands.

2. All lines stacked with the STACK subcommand are stacked FIFO (first in, first out).

3. The length of input lines that are stacked is determined by the current TRUNC setting. The maximum length, however, is 130 characters.

Responses

None. If you issue the STACK subcommand to stack an EDIT subcommand line, the stacked subcommand is executed immediately; responses are those to the stacked subcommands, if any.

TABSET

Use the TABSET subcommand to set logical tab stops for a file. The format of the TABSET subcommand is:

TABSet	n1 [n2 ... nn]

where:

n1 [n2... nn] indicates column positions for logical tab settings. You may specify up to 25 numbers, separated from each other by at least one blank. n1 indicates the first column in the file that may contain data.

Usage Notes

1. The editor assigns the follcwing tab settings by default:

Filetypes	Default Tab Settings
ASM3705, ASSEMBLE, MACRO, UPDATE, UPDTxxxx	1, 10, 16, 31, 36, 41, 46, 69, 72, 80
AMSERV	2, 6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 61, 71, 80
FORTRAN	1, 7, 10, 15, 20, 25, 30, 80
FREEFORT	9, 15, 18, 23, 28, 33, 38, 81
BASIC, VSBASIC	7, 10, 15, 20, 25, 30, 80
PLIOPT, PLI	2, 4, 7, 10, 13, 16, 19, 22, 25, 31, 37, 43, 49, 55, 79, 80
COBOL	1, 8, 12, 20, 28, 36, 44, 68, 72, 80
Others	1, 6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 61, 71, 81, 91, 101, 111, 121, 131

2. Tab setting operands have no effect if the IMAGE subcommand's operand is either OFF or CANON. (CANON is the default for SCRIPT filetypes). A tab entered into a file under these conditions appears as X'05'.

3. The margins set by the TABSET subcommand are used by the INPUT, REPLACE, OVERLAY, and FIND subcommands.

Responses

None.

TOP

Use the TOP subcommand to move the line pointer to the top of the file. The null top line becomes the current line. The format of the TOP subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
| TOP          |                                                       |
└──────────────────────────────────────────────────────────────────────┘
```

Responses

When verification is on, the message:

```
TOF:
```

is displayed.

Display Mode Considerations

When you are using a display terminal, if you specify TOP and verification is on, the current line (see Figure 29) contains the characters TOF (indicating the top of the file), the lines preceding it are blank, and the rest of the screen's output display area contains the first lines of the file.

TRUNC

Use the TRUNC subcommand to change the truncation column of records or to display the current truncation column setting. The format of the TRUNC subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
|              | ┌  ┐                                                  |
| TRUNC        | |n |                                                  |
|              | |* |                                                  |
|              | └  ┘                                                  |
└──────────────────────────────────────────────────────────────────────┘
```

where:

n   indicates the column at which truncation is to occur. If n is specified as an asterisk (*), the truncation column is set to the record length for the filetype.

Usage Notes

1. The editor assigns the following truncation setting by default:

Filetypes	Truncation Column
ASSEMBLE, MACRO, UPDATE, UPDTxxxx	71
AMSERV, COBOL, DIRECT, FORTRAN, PLI, PLIOPT	72
All Others	Record Length

2. The truncation value is used by the INPUT, REPLACE, STACK, and OVERLAY subcommands also, and, for display terminals in display mode, the CHANGE subcommand when it is used with no operands.

3. If your virtual machine is in input mode and you enter a line that is longer than the current truncation setting, the message:

   TRUNCATED

   is displayed along with a display of the truncated line. Your virtual machine is still in input mode.

Responses

When you enter the TRUNC subcommand with no operands, the editor displays the current setting.

TYPE

Use the TYPE subcommand to display all or any part of a file at the terminal. The format of the TYPE subcommand is:

```
┌────────────────────────────────────────────────────────────┐
│              │ ┌   ┌   ┐┐                                   │
│ Type         │ │m  │n  ││                                   │
│              │ │*  │*  ││                                   │
│              │ │1  │   ││                                   │
│              │ └   └   ┘┘                                   │
└────────────────────────────────────────────────────────────┘
```

where:

m indicates the number of lines to be displayed, beginning with the current line. An asterisk (*) indicates all lines between the current line and the end of the file. If m is omitted, only one line is displayed. If the number of lines specified exceeds the number remaining in the file, displaying stops at the end of the file.

n indicates the column at which displaying is to stop, overriding the current end column for verification. If n is specified as an asterisk (*), it indicates that displaying is to take place for the full record length.

Usage Notes

1. Use the TYPE subcommand to display lines when you are editing a file with verification off.

2. If you display one line, the current line pointer does not move; if you display more than one line, the current line is positioned at the last line displayed, or at the end of the file if you specified an asterisk (*).

3. If you have set an end verification column to a value less than the record length, and you want to display an entire record, enter:

   ```
   type 1 *
   ```

4. If you do not specify an end column, the length of the line(s) displayed is determined by the current end verification setting. If you are using right-handed line-number editing on a typewriter terminal or a display terminal in line mode, the line numbers are displayed on the left.

Responses

The requested lines are displayed.

Display Mode Considerations

Since the TYPE subcommand was designed for printing terminals, it is of marginal value on a display terminal, except when you use line mode. However, if the display screen is interrupted by communication from the control program (CP), you should use the TYPE subcommand to restore the full screen display.

UP

Use the UP subcommand to reposition the current line pointer toward the beginning of the file. The format of the UP subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
│              │ ┌   ┐                                                 │
│ Up           │ │n  │                                                 │
│              │ │1  │                                                 │
│              │ └   ┘                                                 │
└──────────────────────────────────────────────────────────────────────┘
```

where:

n indicates the number of lines the pointer is to be moved toward the beginning of the file. If a number is not specified, then the pointer is moved up only one line. The line pointed to becomes the new current line.

Usage Note

UP is equivalent to BACKWARD.

Responses

When verification is on, the line pointed to is displayed at your terminal. If the UP subcommand causes the current line pointer to move beyond the beginning of the file, the following message is displayed:

```
TOF:
```

VERIFY

Use the VERIFY subcommand to set or display the current verificaticn setting. The format of the VERIFY subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
│                                                                      │
│ Verify      │ ┌   ┐ ┌┌         ┐       ┐                             │
│             │ │ON │ ││startcol│ endcol│                             │
│             │ │OFF│ ││   1    │   *   │                             │
│             │ └   ┘ └└         ┘       ┘                             │
└──────────────────────────────────────────────────────────────────────┘
```

where:

ON specifies that lines located, altered, or changed are disFlayed, and changes between edit and input mode are indicated. ON is the initial setting.

OFF specifies that lines that are located, altered, or changed are not disFlayed, and changes between edit and input mode are nct indicated.

startcol indicates the column in which verification is to begin, when verification is on. The default is column 1. startcol must nct be greater than the record length nor greater than endcol.

endcol indicates the last column to be verified, when verification is on. endcol must not be greater than the record length. If endcol is specified as an asterisk (*), each record is disFlayed to the end of the record.

Usage Notes

1. If you issue the VERIFY subcommand with only one operand, that operand is assumed to be the endcol operand. For example, if ycu issue VERIFY 10, verification occurs in columns 1 through 10.

2. The editor assigns the following settings, by default:

Filetypes	Verification End Column
AMSERV, ASSEMBLE, COBOL, DIRECT, FORTRAN, MACRO, PLI, PLIOPT, UPDATE, UPDTxxxx	Column 72
Others (Including FREEFORT)	Record Length

Responses

If you issue the VERIFY subcommand with no operands, the current startcol and endcol settings are displayed, regardless of whether verification is on or off.

X or Y

Use the X or Y subcommands to assign a given EDIT subcommand to be executed whenever X or Y is entered, or to execute the previously assigned subcommand a specified number of times. The format of the X and Y subcommands is:

```
┌──────────────────────────────────────────────────────────────────────┐
│                   │ ┌             ┐                                  │
│  ┌ X ┐            │ │subcommand   │                                  │
│  └ Y ┘            │ │n            │                                  │
│                   │ │1            │                                  │
│                   │ └             ┘                                  │
└──────────────────────────────────────────────────────────────────────┘
```

where:

subcommand indicates any EDIT subcommand line. The editor assumes that you have specified a valid EDIT subcommand, and no error checking is done.

n indicates the number of times the previously assigned subcommand is to be executed. If X or Y is entered with no operands, 1 is assumed.

Usage Notes

1. Advancement of the current line pointer depends upon the EDIT subcommand that has been assigned to X or Y. If a number or a subcommand is not specified, the previously assigned subcommand is executed once.

2. X and Y are initially set to null strings. If you enter X or Y without having previously assigned a subcommand to it, the editor issues the ?EDIT error message.

3. You can use the X and Y subcommands in many instances where you must repeat a subcommand line many times while editing a file, but the situation does not lend itself to a global request. For example, if you assign X to a LOCATE and Y to a CHANGE subcommand, issue:

   ```
   x
   ```

   to execute the LOCATE request, and after examining the line, you can change it and continue searching, by entering the Y subcommand followed by the X subcommand:

   ```
   y#x
   ```

   or just continue searching:

   ```
   x
   ```

Responses

Responses are issued for the EDIT subcommands that are assigned to X and Y, in accordance with the current verification setting.

ZONE

Use the ZONE subcommand to specify the columns of each record (starting position and ending position) to be scanned when the editor searches for a character string or to display the current ZONE settings. The format of the ZONE subcommand is:

```
┌───────────────────────────────────────────────────────────────────────────┐
│                │ ┌          ┌        ┐┐                                   │
│ Zone           │ │firstcol  │lastcol││                                   │
│                │ │*         │*      ││                                   │
│                │ │1         │       ││                                   │
│                │ └          └        ┘┘                                   │
└───────────────────────────────────────────────────────────────────────────┘
```

where:

firstcol indicates the near zone column of each record to be scanned. If firstcol is specified as an asterisk (*), the default is column 1.

lastcol indicates the end zone column of each record to be scanned. If lastcol is specified as an asterisk (*), the default is the reccrd length.

Usage Notes

1. The editor assigns the following settings by default:

Filetypes	Near Zone	End Zone
	(Column)	(Column)
ASSEMBLE, MACRO, UPDATE, UPDTxxxx	1	71
AMSERV, PLI, PLIOPT	2	72
COBOL, DIRECT, FORTRAN	1	72
BASIC, VSBASIC	7	Record Length
FREEFORT	9	Record Length
Others	1	Record Length

2. The ZONE settings are used by the ALTER, CHANGE, and LOCATE subcommands to define the columns that will be scanned. If you specify a character string longer than the zone, you receive the message:

   ZONE ERROR

   and the subcommand is not executed.

3. If you issue a CHANGE subcommand that increases the length of a line beyond the end zone setting, the line is truncated.

4. You can use the ZONE subcommand to protect data in particular columns, for example:

```
    edit newfile memo
    NEW FILE:
    EDIT:
    zcne
       1  80
    zcne 10 20
    input the zcne is now set for columns 10-20

    EDIT:
    change /o/*/
    the zone is n*w set for columns 10-20
```

   Note that the LOCATE and CHANGE subcommands operated on the word now, nct the word zone, because scanning started in position 10, not in pcsition 1.

Responses

When you enter the ZONE subcommand without specifying zone settings, the editor displays the current setting.

? (QUESTION MARK)

Use the ? subcommand to display the last EDIT subcommand executed except for a REUSE (=) or ? (question mark) subcommand. The format of the ? subcommand is:

?	

Usage Note

After an X, Y, or = subcommand, the last EDIT subcommand is the subcommand that was executed as a result of issuing the X or Y subcommand.

Display Mode Considerations

When you issue the ? subcommand using a 3270 in display mode, the last EDIT subcommand that was executed is redisplayed in the user input area. Press the Enter key to execute it again; you may modify the line before reentering it.

nnnnn

Use the nnnnn subcommand to enter and locate lines when you are using line-number editing. The format of the nnnnn subcommand is:

```
┌──────────────────────────────────────────────────────┐
│{nnnnn    } │ [text]                                  │
│{nnnnnnnn } │                                         │
└──────────────────────────────────────────────────────┘
```

__where__:

nnnnn indicates a line number between 0 and 99999 if the filetype is BASIC or VSBASIC, or a line number between 0 and 99999999 if the filetype is FREEFORT.

text specifies a line of text to be inserted into the file at the specified line number. If a line with that number already exists, it is replaced. If no text line is specified, the current line pointer is positioned at the line number specified.

Usage Note

The nnnnn subcommand is valid only when you are using line-number editing; that is, you have issued the LINEMODE subcommand using the RIGHT or LEFT operand. Line-number editing is the default for VSBASIC and FREEFORT files.

Responses

When you issue the nnnnn subcommand with no operands, the line with the specified line number is displayed. If the line is not found, the editor displays the message:

```
LINE NOT FOUND
```

and the current line pointer is set at the next line number greater than nnnnn.

EDIT Macros

Edit macros are CMS EXEC files that execute sequences of EDIT subcommands. The following edit macros are supplied with VM/370 for your convenience. For additional information on creating and invoking your own edit macros and EXEC files, see the VM/370 CMS User's Guide.

$DUP

Use the $DUP to duplicate the current line. The format of the $DUP macro is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│         │ ┌   ┐                                                          │
│  $DUP   │ │n  │                                                          │
│         │ │1  │                                                          │
│         │ └   ┘                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```

where:

n   indicates the number of times you want to duplicate the line; the maximum value you can specify is 25. If n is omitted, the current line is duplicated once.

Usage Notes

1. The last copy of the line duplicated becomes the new current line.

2. If you use the logical line end symbol (#) to stack additional subcommands on the same line with the $DUP edit macro those subcommands are cleared from the console stack and the message:

   ```
   STACKED LINES CLEARED BY $DUP
   ```

   is issued. The stacked subcommand(s) are not executed.

3. Because it uses console functions, $DUP cannot be used when duplicating records containing binary zeros or nonprintable characters. Truncated duplicate records will result.

4. When using line-number editing, you can insert duplicate lines between existing numbered lines if the interval between line numbers is large enough. Execution of $DUP stops after the last valid line number has been assigned. You can renumber your file to increase the interval between line numbers.

Responses

The last line duplicated (the new current line) is displayed.

$MOVE

Use the $MOVE edit macro to move one or more lines from one place in a file to another place. The format of the $MOVE macro is:

```
| $MOVE  |  n (UP m     )
|        |    <DOWN m    >
|        |    (TO label  )
```

where:

n indicates the number of records you want to move, beginning with the current line. The maximum number of lines you can move is 25.

UP m indicates that you want to move the lines toward the top of the file, m lines above the current line.

DOWN m indicates that you want to move the lines toward the end of the file, m lines below the last line you are going to move.

TO label indicates that you want the lines inserted following the specified label. The label must be one to eight uppercase characters and must start in column 1.

Usage Notes

1. The last line moved becomes the new current line.

2. If the label is not found or if the DOWN value exceeds the number of lines remaining before end of file, the lines are inserted at the end of the file. If the UP value exceeds the number of lines remaining before top of file, the lines are inserted at the top of the file.

3. If you use the logical line end symbol (#) to stack additional subcommands on the same line with the $MOVE request, those subcommands are cleared from the console stack and the message:

   ```
   STACKED LINES CLEARED BY $MOVE
   ```

   is displayed. The stacked subcommands are not executed.

4. Because it uses console functions, $MOVE will truncate duplicated records containing binary zeros or nonprintable characters.

Responses

When verification is on, the last line moved is displayed.

Section 4. DEBUG Subcommands

This section describes the subcommands that are available to you when you use the debug environment to test and debug your programs. The debug environment is entered when:

- The DEBUG command is issued from the CMS environment. (The DEBUG command is described in "Section 2. CMS Commands.")
- An external interruption occurs. (An external interruption is caused by the CP EXTERNAL command.)
- A breakpoint (instruction address stop) is encountered during program execution. (Breakpoints are set with the DEBUG subcommand BREAK.)

When the debug environment is entered, the contents of all general registers, the channel status word (CSW), and the channel address word (CAW) are saved so they may be examined and changed before being restored when leaving the debug environment. If debug is entered via an interruption, the old program status word (PSW) for that interruption is also saved. If DEBUG is the first command entered after an abnormal termination (abend) occurs, the contents of all general registers, the CSW, the CAW, and the old PSW are available from the time of the abend.

For hints on debugging your programs using the CMS debug environment, consult the *VM/370 CMS User's Guide*.

BREAK

Use the BREAK subcommand to stop execution of a program or module at a specific instruction location called a breakpoint. The format of the BREAK subcommand is:

```
┌─────────────────────────────────────────────────────────────────────┐
│ BReak   │  id {symbol}                                              │
│         │     {hexloc}                                              │
└─────────────────────────────────────────────────────────────────────┘
```

where:

id   is a decimal number, from 0 to 15, which identifies the breakpoint. A maximum of 16 breakpoints may be in effect at one time; if you specify an identification number that is already set for a breakpoint, the previous breakpoint is cleared and the new one is set.

symbol   is a name assigned to the storage location where the breakpoint is set. symbol, if used, must have previously been set using the DEFINE subcommand.

hexloc   is the hexadecimal storage location (relative to the current origin) where the breakpoint is to occur. hexloc must be on a halfword boundary and its value added to the current origin must not exceed your virtual machine size.

Usage Notes

1. To set breakpoints before beginning program execution, enter the debug environment with the DEBUG command after you load the program into storage. After setting the breakpoints, use the RETURN subcommand to leave the debug environment and issue the START command to begin program execution. For example:

   ```
   load myprog
   debug
   break 1 20016
   break 2 20032
   return
   start
   ```

2. When you assign hexloc to a breakpoint, you must know the current origin (set with the ORIGIN subcommand). The hexloc you specify is added to the current origin to determine the breakpoint address.

3. When a breakpoint is found during program execution, the message:

   ```
   DMSDBG728I DEBUG ENTERED BREAKPOINT yy AT xxxxxx
   ```

   is displayed at the terminal. To resume program execution, use the GO subcommand.

4. Breakpcints are cleared after they are encountered; thus, if a breakpoint is encountered during a program loop you must reset the breakpoint if you want to interrupt execution the next time that address is encountered.

5. When you set a breakpoint, the halfword at the address specified is replaced with B2Ex, where x represents the identification number you assigned. After the breakpoint is encountered during execution, B2Ex is replaced with the original operation code.

6. You should set breakpoints only at valid operation code addresses; the BREAK subcommand does not check to see whether or not the specified location contains a valid operation code.

7. If you reference a virtual storage address that is in a shared segment, you are given a nonshared copy of the segment and you receive the message:

```
SYSTEM sysname REPLACED WITH NON-SHARED copy
```

Responses

None.

CAW

Use the CAW subcommand to display at the terminal the contents of the CAW (channel address word) as it existed at the time the debug environment was entered. The format of the CAW subcommand is:

```
┌──────────────────────────────────────────────────────────────┐
|  CAW |                                                       |
└──────────────────────────────────────────────────────────────┘
```

Usage Notes

1. Issue the CAW subcommand to check that the command address field contains a valid CCW address, or to find the address of the current CCW so you can examine it.

2. The three low-order bits of the command address field must be zeros in order for the CCW to be on a doubleword boundary. If the CCW is not on a doubleword boundary or if the command address specifies a location protected from fetching or outside the storage of a particular user, the Start I/O instruction causes the status portion of the CSW (channel status word) to be stored with the program check or protection check bit on. In this event, the I/O operation is not initiated.

Responses

The CAW, located at storage location X'48', is displayed. Its format is:

```
┌──────────────────────────────────────────────────────────────┐
| KEY | 0000 |  Command Address                                |
└──────────────────────────────────────────────────────────────┘
0    3 4    7 8                                               31
```

Bits Contents

0-3 The protection key for all commands associated with Start I/O. The protection key in the CAW is compared to a key in storage whenever a reference is made to storage.

4-7 This field is not used and must contain binary zeros.

8-31 The command address field contains the storage address (in hexadecimal representation) of the first CCW (channel command word) associated with the next or most recent Start I/O.

CSW

Use the CSW subcommand to display at the terminal the contents of the CSW (channel status word), as it existed at the time the debug environment was entered. The format of the CSW subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
|  CSW  |                                                              |
└──────────────────────────────────────────────────────────────────────┘
```

Usage Notes

1. The CSW indicates the status of the channel or an input/output device, or the conditions under which an I/O operation terminated. The CSW is formed in the channel and stored in storage location X'40' when an I/O interruption occurs. If I/O interruptions are suppressed, the CSW is stored when the next Start I/O, Test I/O, or Halt I/O instruction is executed.

2. Whenever an I/O operation abnormally terminates, issue the CSW subcommand. The status and residual count information in the CSW is very useful in debugging. Also, use the CSW to calculate the address of the last executed CCW (subtract eight bytes from the command address to find the address of the last CCW executed).

Responses

The contents of the CSW are displayed at the terminal in hexadecimal representation. Its format is:

```
┌──────────────────────────────────────────────────────────────────────┐
|KEY|0000|   Command Address   |  Status   |  Byte Count               |
└──────────────────────────────────────────────────────────────────────┘
0  3 4  7 8                  31 32       47 48                        63
```

Bits — Contents

0-3 The protection key is moved to the CSW from the CAW. It shows the protection key at the time the I/O operation started. The contents of this field are not affected by programming errors detected by the channel or by the condition causing termination of the operation.

4-7 This field is not used and must contain binary zeros.

8-31 The command address contains a storage address (in hexadecimal representation) that is eight bytes greater than the address of the last CCW executed.

32-47 The status bits indicate the conditions in the device or channel that caused the CSW to be stored.

48-63 The residual count is the difference between the number of bytes specified in the last executed CCW and the number of bytes that were actually transferred. When an input operation is terminated, the difference between the original count in the CCW and the residual count in the CSW is equal to the number of bytes transferred to storage; on an output operation, the difference is equal to the number of bytes transferred to the I/O device.

DEFINE

Use the DEFINE subcommand to assign a symbolic name to a specific storage address. Once a symbolic name is assigned to a storage address, that symbolic name can be used to refer to that address in any of the other DEBUG subcommands. The format of the DEFINE subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
│          │                      ┌          ┐                          │
│ DEFine   │ symbol   hexloc      │bytecount│                          │
│          │                      │    4     │                          │
│          │                      └          ┘                          │
└──────────────────────────────────────────────────────────────────────┘
```

where:

symbol is the name to be assigned to the storage address derived from the second operand, hexloc. Symbol may be from one to eight characters long, and must contain at least one nonhexadecimal character. Any symbolic name longer than eight characters is left-justified and truncated on the right after the eighth character.

hexloc is the hexadecimal storage location, in relation to the current origin, to which the name specified in the first operand (symbol), is assigned.

bytecount
is a decimal number, between 1 and 56 inclusive, which specifies the length in bytes of the field whose name is specifed by the first operand (symbol) and whose starting location is specified by the second operand (hexloc). When bytecount is not specified, 4 is assumed.

Usage Notes

1. Issuing the DEFINE subcommand creates an entry in the debug symbol table. The entry consists of the symbol name, the storage address, and the length of the field. A maximum of 16 symbols can be defined in the debug symbol table at any given time.

2. When a DEFINE subcommand specifies a symbol that already exists in the debug symbol table, the storage address derived from the current request replaces the previous storage address. Several symbols may be assigned to the same storage address, but each of these symbols constitutes one entry in the debug symbol table. The symbols remain defined until they are redefined or until an IPL subcommand loads a new copy of CMS.

3. When you assign a symbolic name to a storage location, you must know the current origin (set by the ORIGIN subcommand). The hexloc you specify is added to the current origin to create the entry in the symbol table used by DEBUG subcommands. If you change the current origin, existing entries are not changed.

4. You can use symbolic names to refer to storage locations when you issue the DEBUG subcommands BREAK, DUMP, GO, ORIGIN, STORE, and X.

Responses

None.

DUMP

Use the DUMP subcommand to print part or all of your virtual storage on the printer. The requested information is printed offline as soon as the printer is available. First, a heading:

ident FROM starting location TO ending location

is printed. Next, the general registers 0-7 and 8-15, and the floating-point registers 0-6 are printed, followed by the PSW, CSW, and CAW. Then the specified portion of virtual storage is printed with the storage address of the first byte in the line printed at the left, followed by the alphameric interpretation of 32 bytes of storage. The format of the DUMP subcommand is:

```
┌─────────────────────────────────────────────────────────────────────┐
│          │ ┌              ┌                          ┐  ┐           │
│  DUmp    │ │ symbol1      │ symbol2                  │  │           │
│          │ │ hexloc1      │ hexloc2    [ident]       │  │           │
│          │ │    0         │    *                     │  │           │
│          │ │              │   32                     │  │           │
│          │ └              └                          ┘  ┘           │
└─────────────────────────────────────────────────────────────────────┘
```

where:

symbol1 is the name assigned (via the DEFINE subcommand) to the storage address that begins the dump.

hexloc1 is the hexadecimal storage location, in relation to current origin, that begins the dump.

symbol2 is the name assigned (via the DEFINE subcommand) to the storage address that ends the dump.

hexloc2 is the hexadecimal storage location, in relation to the current origin, that ends the dump.

* indicates that the dump ends at your virtual machine's last virtual storage address.

ident is any name (up to eight characters) that identifies the dump.

Usage Notes

1. If you issue the DUMP subcommand with no operands, 32 bytes of storage are dumped, starting at the current origin.

2. The first and second operands must designate storage addresses that do not exceed your virtual machine storage size. Also, the storage address derived from the second operand must be greater than the storage address derived from the first operand.

Responses

None.

GO

Use the GO subcommand to exit from the debug environment and begin program execution. The format of the GO subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
│       │  ┌          ┐                                                │
│  GO   │  │ symbol   │                                                │
│       │  │ hexloc   │                                                │
│       │  └          ┘                                                │
└──────────────────────────────────────────────────────────────────────┘
```

where:

symbol is the symbolic name assigned to the storage location where you want execution to begin.

hexloc is the hexadecimal location, in relation to the current origin, where you want execution to begin.

Usage Notes

1. When you issue the GO subcommand, the general registers, CAW (channel address word), and CSW (channel status word) are restored either to their contents upon entering the debug environment, or, if they have been modified, to their modified contents. Then the old PSW is loaded and becomes the current PSW. Execution begins at the instruction address contained in bits 40-63 of the PSW.

2. When you specify symbol or hexloc with the GO subcommand, the specified address replaces the instruction address in the old PSW, so execution will begin at that address. If you entered the debug environment with the DEBUG command, you must specify an address with the GO subcommand.

3. The address you specify must be within your virtual machine and it must contain a valid operation code.

Responses

Program execution is resumed.

GPR

Use the GPR subcommand to display the contents of one or more general registers at the terminal. The format of the GPR subcommand is:

```
| GPR | reg1  [reg2]                                       |
```

where:

reg1 is a decimal number (from 0-15 inclusive) indicating the first or only general register whose contents are to be displayed.

reg2 is a decimal number (from 0-15 inclusive) indicating the last general register whose contents are to be displayed. reg2 must be larger than reg1.

Responses

The register or registers specified are displayed, in hexadecimal representation:

```
xxxxxxxx
  .
  .
  .
```

HX

Use the HX subcommand to leave the debug environment, regardless of the reason the debug environment was entered. The format of the HX subcommand is:

```
|   HX   |                                                  |
```

Responses

If you entered the debug environment following a program interruption, you receive the message:

```
CMS
```

to indicate a return to the CMS environment. If you entered the debug environment by issuing the DEBUG command, you receive the message:

```
DMSABN148T SYSTEM ABEND 2E4 CALLED FROM xxxxxx
```

where xxxxxx is the address of the debug routine.

ORIGIN

Use the ORIGIN subcommand to set an origin or base address to be used in the debug environment. The format of the ORIGIN subcommand is:

```
┌──────────────────────────────────────────────────────────┐
│ ORigin │ ⎧symbol⎫                                          │
│        │ ⎨hexloc⎬                                          │
│        │ ⎩  0   ⎭                                          │
└──────────────────────────────────────────────────────────┘
```

where:

symbol is a symbolic name that was previously assigned (via the DEFINE subcommand) to a storage address.

hexloc is a hexadecimal location within the limits of your virtual storage. If you do not explicitly set an origin, then it has a value of 0.

Usage Notes

1. When the ORIGIN subcommand specifies a symbol, the debug symbol table is searched. If a match is found, the value corresponding to the symbol becomes the new origin. When a hexadecimal location is specified, that value becomes the origin. In either case, the operand cannot specify an address greater than your virtual storage size.

2. Any origin set by an ORIGIN subcommand remains in effect until another ORIGIN subcommand is issued, or until you obtain a new copy of CMS. Whenever a new ORIGIN subcommand is issued, the value specified in that subcommand overlays the previous origin setting. If you obtain a new copy of CMS (via IPL), the origin is set to 0 until a new ORIGIN subcommand is issued.

3. You can use the ORIGIN subcommand to set the origin to your program's base address, and then refer to actual instruction addresses in your program, rather than to virtual storage locations.

Responses

None.

PSW

Use the PSW subcommand to display the contents of the PSW (program status word). The format of the PSW subcommand is:

```
| PSW |
```

Usage Notes

1. If the debug environment was entered because of a program interruption, the program old PSW is displayed. If the debug environment was entered because of an external interruption, the external old PSW is displayed. If the debug environment was entered for any other reason, the following is displayed in response to the PSW subcommand:

   01000000xxxxxxxx

   where the 1 in the first byte means that external interruptions are allowed and xxxxxxxx is the hexadecimal storage address of the debug program.

2. The PSW contains some information not contained in storage or registers but required for proper program execution. In general, the PSW is used to control instruction sequencing and to hold and indicate the status of the system in relation to the program currently executing. For a description of the PSW, refer to "Appendix A: System/370 Information" in the VM/370 System Programmer's Guide.

Responses

The PSW is displayed in hexadecimal representation.

RETURN

Use the RETURN subcommand to exit from the debug environment and enter the CMS command environment. The format of the RETURN subcommand is:

```
| RETurn |
```

Usage Note

The RETURN subcommand is valid only when the debug environment was entered via the DEBUG command.

Responses

The CMS ready message indicates that control has been returned to the CMS environment.

SET

Use the SET subcommand to change the contents of the control words and general registers. The format of the SET subcommand is:

```
| SET | (CAW  hexinfo                              ) |
|     | )CSW  hexinfo  [hexinfo]                   ( |
|     | )PSW  hexinfo  [hexinfo]                   ( |
|     | (GPR  reg        hexinfo   [hexinfo]       ) |
```

where:

CAW hexinfo
stores the specified information (hexinfo) in the CAW (channel address word) that existed at the time the debug environment was entered.

CSW hexinfo [hexinfo]
stores the specified information (hexinfo [hexinfo]) in the CSW (channel status word) that existed at the time the debug environment was entered.

PSW hexinfo [hexinfo]
stores the specified information (hexinfo [hexinfo]) in the old PSW (program status word) for the interruption that caused the debug environment to be entered.

GPR reg hexinfo [hexinfo]
stores the specified information (hexinfo [hexinfo]) in the specified general register (reg).

Usage Notes

1. The SET subcommand can only change the contents of one control word at a time. For example, you must issue the SET subcommand three times:

   ```
   set caw hexinfo
   set csw hexinfo [hexinfo]
   set psw hexinfo [hexinfo]
   ```

   to change the contents of the three control words.

2. The SET subcommand can change the contents of one or two general registers each time it is issued. When four or fewer bytes of information are specified, only the contents of the specified register are changed. When more than four bytes of information are specified, the contents of the specified register and the next sequential register are changed. For example, the SET subcommand:

   ```
   set gpr 2 xxxxxxxx
   ```

   changes only the contents of general register 2. But, the SET subcommand:

   ```
   set gpr 2 xxxxxxxx xxxxxxxx
   ```

   changes the contents of general registers 2 and 3.

3. Each hexinfo operand should be from one to four bytes long. If an operand is less than four bytes and contains an uneven number of hexadecimal digits (representing half-byte information), the information is right-justified and the left half of the uneven byte is set to zero. If more than eight hexadecimal digits are specified in a single operand, the information is left-justified and truncated on the right after the eighth digit.

4. The number of bytes that can be stored using the SET subcommand varies depending on the form of the subcommand. With the CAW form, up to four bytes of information may be stored. With the CSW, GPR, and PSW forms, up to eight bytes of information may be stored, but these bytes must be represented in two operands of four bytes each. When two operands of information are specified, the information is stored in consecutive locations (or registers), even if one or both operands contain less than four bytes of information.

Responses

None. To display the contents of control words or registers after you modify them, you must use the CAW, CSW, PSW, and GPR subcommands.

STORE

Use the STORE subcommand to store up to 12 bytes of hexadecimal information in any valid virtual storage location. The format of the STORE subcommand is:

```
| STore | {symbol} hexinfo [hexinfo [hexinfo]]    |
|       | {hexloc}                                |
```

where:

symbol is the symbolic name assigned (via the DEFINE subcommand) to the storage address where the first byte of specified information is to be stored.

hexloc is the hexadecimal location, relative to the current origin, where the first byte of information is to be stored.

hexinfo is the hexadecimal information, four bytes or less in length (that is, two to eight hexadecimal digits), to be stored.

Usage Notes

1. If an operand is less than four bytes long and contains an uneven number of hexadecimal digits (representing half-byte information), the information is right-justified and the left half of the uneven byte is set to zero. If more than eight hexadecimal digits are specified in a single operand, the information is left-justified and truncated on the right after the eighth digit.

2. The STORE subcommand can store a maximum of 12 bytes at one time. By specifying all three information operands, each containing four bytes of information, the maximum 12 bytes can be stored. If less than four bytes are specified in any or all of the operands, the information given is arranged into a string of consecutive bytes, and that string is stored starting at the location derived from the first operand.

For example, if you have defined a four-byte symbol named FENCE that currently contains X'FFFFFFFF' and you enter:

```
store fence 0
```

FENCE contains X'00FFFFFF'.

Responses

None. To display the contents of a storage location after you have modified it, you must use the X subcommand.

X

Use the X subcommand to examine and display the contents of specific locations in virtual storage. The format of the X (examine) subcommand is:

```
┌──────────────────────────────────────────────────────┐
│ X │ ⎧ symbol   ┌        ┐ ⎫                            │
│   │ ⎪          │ n      │ ⎪                            │
│   │ ⎨          │ length │ ⎬                            │
│   │ ⎪          └        ┘ ⎪                            │
│   │ ⎪ hexloc   ┌        ┐ ⎪                            │
│   │ ⎪          │ n      │ ⎪                            │
│   │ ⎩          │ 4      │ ⎭                            │
│   │            └        ┘                              │
└──────────────────────────────────────────────────────┘
```

where:

symbol n is the name assigned (via the DEFINE subcommand) to the storage address of the first byte to be displayed. n is a decimal number from 1 to 56 inclusive, that specifies the number of bytes to be examined. If a symbol is specified without a second operand, the length attribute associated with that symbol in the debug symbol table specifies the number cf bytes to be examined.

hexloc n is the hexadecimal location, in relation to the current origin, of the first byte to be examined. If hexloc is specified without a second operand, four bytes are displayed.

Usage Note

The address represented by symbol or hexloc must be within your virtual machine storage size.

Responses

The requested information is displayed at the terminal in hexadecimal format.

Section 5. EXEC Control Statements

This section describes the formats, usage rules, and default values for EXEC control words, including:

- Control statements
- Built-in functions
- Special variables

An EXEC procedure is a CMS file that contains a sequence of CMS commands and/or EXEC control statements. Control statements determine the logic flow for EXEC, provide terminal communications, and may be used to manipulate CMS disk files. For an introduction to the EXEC facilities, and for complete tutorial information, including examples, consult the *VM/370 CMS User's Guide*.

EXEC procedures may be invoked with the EXEC command, described in "Section 2. CMS Commands." You may also execute an EXEC procedure by specifying its filename, as long as the implied EXEC function is in effect.

The Assignment Statement

Use the assignment statement in an EXEC procedure to assign a value to a variable symbol. Variable symbols may be tested and manipulated to control the execution of an EXEC procedure. The format of the assignment statement is:

```
&variable =  { string   }
             { ae       }
             { function }
             { X'xxxxxx }
```

where:

&variable indicates the variable symbol which is assigned the specified value. A variable may contain a maximum of eight alphameric characters, including the initial ampersand, which is required. Except in the EXEC special variables &* and &DISK*, a variable must not contain any special characters.

string is a data item of up to eight characters. It may also be a variable symbol or null. Whether a numeric string is treated as numeric or character data depends on how it is used in the EXEC. If a string containing variable symbols expands to more than eight characters, it is truncated. If the string consists of eight X'FF' characters, the variable is set to a null string.

ae is an arithmetic expression consisting of a sequence of data items that possess positive or negative integral values and are separated by plus or minus signs:

```
&1 - 4 + &CALC - 6
```

function is an EXEC built-in function followed by at least one token.

X'xxxxxx indicates up to six hexadecimal digits to be converted to decimal before assignment. For example:

```
&A = X'C0
```

results in &A having the decimal value 192.

Hexadecimal conversion is not performed unless you have used the &HEX ON control statement.

Variable Substitution

All variable symbols occurring in executable statements are substituted before the statement is executed. An executable statement is (1) a CMS command line, or (2) an EXEC control statement (including assignment statements).

Variable substitution is performed on all symbols on the left-hand side of an assignment statement, except the leftmost variable. For example:

```
            &I = 2
            &X&I = 5
```

sets &X2 to 5.

If a variable on the left-hand side of an assignment statement has already been assigned a value, it is replaced by the new value specified in the assignment statement.

If the special form, X'&symbol, is used, the specified symbol is converted to its hexadecimal equivalent. For example:

```
     &A = 192
     &TYPE X'&A
```

results in the display:

```
     C0
```

If a variable symbol that has not been defined is used in an executable statement the symbol is set to a null token and ignored. In some instances this may cause an EXEC processing error.

Tokens

All executable statements in an EXEC are scanned into eight-character tokens, and padded or truncated as necessary. Tokens are formed of words delimited by blanks and parentheses. If there is no blank before or after a parenthesis, one is added in either case. If more than one blank separates a word or a parenthesis from another, the extra blanks are removed from the line. For example, the line:

```
     &TYPE  THIS IS AN EXAGGERATED (MESSAGE
```

scans as:

```
     &TYPE THIS IS AN EXAGGERA ( MESSAGE
```

Variable symbols are substituted after each line is scanned, and each token is scanned repeatedly until all symbols in it are substituted.

In an executable statement, a token beginning with the character X'FF' (or a variable to which such a token is assigned as a value) usually prevents the processing of data following it on the same line. However, if an assignment statement sets a variable to eight X'FF' characters, data following the variable in an executable statement is processed.

&ARGS

Use the &ARGS control statement to redefine the value of one or more of the special variables, &1 through &30. The format of the &ARGS control statement is:

```
| &ARGS | [arg1 [arg2 ... [arg30] ] ]                                 |
```

where:

[arg1 [arg2 ... [arg30]]]
: specify up to 30 tokens to be assigned to the special variables &1 through &30. If no arguments are specified, all of the variables &1 through &30 are set to blanks. When fewer than 30 arguments are entered, the remaining arguments are set to blanks. An argument is also set to blanks if it is specified as a percent sign (%).

Usage Notes

1. To enter an argument list from the terminal, use the &READ ARGS control statement.
2. An &ARGS control statement resets the values of the &INDEX, &*, and &$ special variables.

&BEGEMSG

Use the &BEGEMSG control statement to introduce one or more unscanned lines to be edited as VM/370 error messages. The list of lines to be displayed must be terminated by an &END control statement, which must appear beginning in column 1. The format of the &BEGEMSG control statement is:

```
| &BEGEMSG | [ALL]                                                  |
```

where:

ALL specifies, for fixed-length EXEC files, that the entire line (to a maximum of 130 characters) is to be displayed.

Usage Notes

1. To qualify for error message editing, the first data item on each line following the &BEGEMSG control statement must be seven characters long, in the format:

   mmmnnns

   where:

   mmmnnn is a six-character message identification you can supply for the error message. Standard VM/370 error messages use a three-character module code (mmm) and a three-character message number (nnn).

   s indicates the severity code. The following codes qualify the message for error message editing:

Code	Message Type
I	Informational
E	Error
W	Warning

   When the severity code is E, I, or W, the message is displayed in accordance with the CP EMSG setting (ON, OFF, CODE, or TEXT). You can change this setting with the CP SET command, described in *VM/370 CP Command Reference for General Users*.

2. When you use the &BEGEMSG control statement to display error messages, the character string "DMS" is inserted in front of the seven-character message identification. For example, if the EMSG setting is ON, the lines:

   ```
   &BEGEMSG
   TEST01E INSURMOUNTABLE ERROR
   &END
   ```

   result in the display:

   ```
   DMSTEST01E INSURMOUNTABLE ERROR
   ```

   Note: Since the maximum length of a line that you can display at your terminal is 130 characters, the insertion of the characters DMS will cause lines greater than 127 characters long to be truncated.

3. Messages that are displayed as the result of an &BEGEMSG control statement are not scanned by the EXEC interpreter. Therefore, no variable substitution is performed and no data items are truncated. To display variable data, use the &EMSG control statement.

&BEGPUNCH

Use the &BEGPUNCH control statement to delimit the beginning of a list of one or more data lines to be spooled to your virtual card punch. The list of lines to be punched is terminated by the control statement &END, which must occur beginning in column 1. The format of the &BEGPUNCH control statement is:

```
| &BEGPUNCH  | [ALL]                                                    |
```

where:

ALL specifies that data occupying columns 73 through 80 should be punched. If ALL is not specified, input records are truncated at column 72 and columns 73 through 80 of the output record are padded with blanks.

Usage Notes

1. Lines that are punched as the result of an &BEGPUNCH control statement are not scanned by the EXEC interpreter. Therefore, no variable substitution is performed and no data items are truncated. To punch variable data, you must use the &PUNCH control statement.

2. When you are finished punching lines in an EXEC procedure, you should use the CP CLOSE command to close your virtual punch.

&BEGSTACK

Use the &BEGSTACK control statement to delimit the beginning of a list of one or more data lines to be placed in the console input stack. The list of lines to be stacked is terminated by the control statement &END which must occur beginning in column 1. The format of the &BEGSTACK control statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
│              │  ┌    ┐  ┌   ┐                                        │
│  &BEGSTACK   │  │FIFO│  │ALL│                                        │
│              │  │LIFO│  └   ┘                                        │
│              │  └    ┘                                               │
└──────────────────────────────────────────────────────────────────────┘
```

<u>where</u>:

<u>FIFO</u> specifies that the lines that follow are to be stacked on a first in, first out basis. This is the default value.

LIFO specifies that the lines that follow are to be stacked on a last in, first out basis.

ALL specifies, for fixed-length EXEC files, that the entire line (to a maximum of 130 characters) is to be stacked. If ALL is not specified, the lines are truncated in column 72.

<u>Usage Notes</u>

1. Lines that are stacked as the result of an &BEGSTACK control statement are not scanned by the EXEC interpreter. Therefore, no variable substitution is performed, and data items are not truncated. To stack variable data, you must use the &STACK control statement.

2. To stack a null line in an EXEC file you must use the &STACK control statement. A null line following an &BEGSTACK control statement is interpreted as a line of blanks. To stack an INPUT, REPLACE, or CHANGE subcommand to enter input mode from a fixed-length EXEC, you should use the &STACK control statement.

&BEGTYPE

Use the &BEGTYPE control statement to delimit the beginning of a list of one or more data lines to be displayed at the terminal. The list of lines to be displayed is terminated by the control statement &END, which must occur beginning in column 1. The format of the &BEGTYPE control statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
│  &BEGTYPE   │  [ALL]                                                 │
└──────────────────────────────────────────────────────────────────────┘
```

<u>where</u>:

ALL specifies, for fixed-length EXEC files, that data occupying columns 73 through 130 is to be displayed. If ALL is nct specified, the lines are truncated at column 72.

Usage Note

Lines that are displayed as the result of an &BEGTYPE control statement are not scanned by the EXEC interpreter. Therefore, no variable substitution is performed, and data items are not truncated. To display variable data, you must use the &TYPE control statement.

&CONTINUE

Use the &CONTINUE control statement to instruct the EXEC interpreter to process the next statement in the EXEC file. The format of the &CONTINUE control statement is:

```
| &CONTINUE |                                                    |
```

Usage Note

&CONTINUE is generally used with an EXEC label (for example, -LAB &CONTINUE) to provide a branch address for &ERROR, &GOTO, and other branching statements. &CONTINUE is the default action taken when an error is detected in processing a CMS command.

&CONTROL

Use the &CONTROL control statement to specify the amount of data to be displayed in the execution summary of an EXEC. The format of the &CONTROL control statement is:

```
┌──────────────────────────────────────────────────────────────────┐
│                   │                                              │
│   &CONTROL        │  ┌      ┐  ┌      ┐  ┌       ┐  ┌       ┐    │
│                   │  │OFF   │  │MSG   │  │TIME   │  │PACK   │    │
│                   │  │ERROR │  │NOMSG │  │NOTIME │  │NOPACK │    │
│                   │  │CMS   │  └      ┘  └       ┘  └       ┘    │
│                   │  │ALL   │                                    │
│                   │  └      ┘                                    │
└──────────────────────────────────────────────────────────────────┘
```

where:

OFF suppresses the display of CMS commands and EXEC control statements as they execute and of any return codes that may result from CMS commands.

ERROR displays only those CMS commands that result in an error and also displays the error message and the return code.

CMS displays each CMS command as it is executed and all nonzero return codes.

ALL displays CMS commands and EXEC executable statements as they execute as well as any nonzero return codes from CMS commands.

MSG does not suppress the "FILE NOT FOUND" message if it is issued by the following commands when they are invoked from an EXEC procedure: ERASE, LISTFILE, RENAME, or STATE.

NOMSG suppresses the "FILE NOT FOUND" message if it is issued when the ERASE, LISTFILE, RENAME, or STATE commands are invoked from an EXEC procedure.

TIME includes the time-of-day value with each CMS command printed in the execution summary; for example:

```
14:36:30 TYPE A B
```

This operand is effective only if CMS or ALL is also specified.

NOTIME does not include the time-of-day value with CMS commands printed in the execution summary.

PACK packs the lines of the execution summary so that surplus blanks are removed from the displayed lines.

NOPACK does not pack the lines of the execution summary.

Usage Notes

1. The execution summary may consist of CMS commands, responses, errcr messages, and return codes, as well as EXEC control statements and assignment statements. When EXEC statements are displayed, they are displayed in their scanned format, with all variable symbols substituted.

2. Each operand remains set until explicitly reset by another &CONTRCL statement that specifies a conflicting operand. When &CONTROL is used with no operands, all operands are reset to their default values.

3. There is no global setting for &CONTROL. When an EXEC is nested within another EXEC, the execution summary is controlled ty the nested EXEC's &CONTROL setting. When control returns to the outer EXEC, the original &CONTROL setting is restored.

&EMSG

Use the &EMSG control statement to display a line of tokens to be edited as a VM/370 error message. The format of the &EMSG control statement is:

```
| &EMSG | mmmnnns [tok1 ... [tokn]]                                |
```

where:

mmmnnn is a six-character identification you may supply for the error message. Standard VM/370 messages are coded using a three-character module code (mmm) and a three-character message number (nnn).

s indicates the severity code. The following codes qualify the message for error message editing:

Code	Message Type
I	Information
E	Error
W	Warning

tok1 ...[tokn]
is the text of the message to be displayed.

Usage Notes

1. When the severity code is I, E, or W, the message is displayed in accordance with the CP EMSG setting (ON, OFF, CODE, or TEXT). You can change the setting with the CP SET command, described in VM/370 CP Command Reference for General Users.

2. When an &EMSG code is displayed, it is prefixed with DMS. Fcr example, the statement:

   ```
   &EMSG ERROR1E INVALID ARGUMENT
   ```

   displays as follows when the EMSG setting is ON:

   ```
   DMSERROR1E INVALID ARGUMENT
   ```

3. To display an error message with unsubstituted data, or to display a line with words of more than eight characters, use the &BEGEMSG control statement.

&END

Use the &END control statement to terminate a list of one or more lines that began with an &BEGEMSG, &BEGPUNCH, &BEGSTACK, or &BEGTYPE contrcl statement. The format of the &END control statement is:

```
┌──────────────────────────────────────────────────────────────┐
│  &END  │                                                      │
└──────────────────────────────────────────────────────────────┘
```

The word "&END" must be entered beginning in column 1.

&ERROR

Use the &ERROR control statement to specify the action to be taken when a CMS command results in an error and returns with a nonzero return code. The format of the &ERROR control statement is:

```
┌──────────────────────────────────────────────────────────────┐
│          │  ┌                    ┐                            │
│  &ERROR  │  │executable-statement│                            │
│          │  │&CONTINUE           │                            │
│          │  └                    ┘                            │
└──────────────────────────────────────────────────────────────┘
```

where:

executable-statement
specifies any executable statement, which may be an EXEC control statement or assignment statement or a CMS command. If you specify an EXEC control statement that transfers control to another line in the EXEC, execution continues at the specified line. Otherwise, execution continues with the line following the CMS command line that caused the error.

Usage Notes

1. If your EXEC does not contain an &ERROR control statement, then the default is &CONTINUE; that is, EXEC processing is to continue with the line following the CMS command that caused the error. You can use &ERROR &CONTINUE to reset a previous &ERROR statement.

2. The words following an &ERROR control statement are not scanned until a CMS command returns a nonzero return code. Therefore, if you specify an invalid EXEC statement, the error is not detected until a CMS command failure triggers the &ERROR statement. If the &ERROR statement executes a CMS command that also results in an error, EXEC processing is terminated.

&EXIT

Use the &EXIT control statement to terminate processing the EXEC file. If the exit is taken from a first-level EXEC procedure, control passes to CMS. If the exit is taken from a nested EXEC procedure, control passes to the calling EXEC procedure. The format of the &EXIT contrcl statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
│         │  ┌           ┐                                               │
│  &EXIT  │  │return-code│                                               │
│         │  │     0     │                                               │
│         │  └           ┘                                               │
└──────────────────────────────────────────────────────────────────────┘
```

where:

return-code
: specifies a numeric value, which may be a variable symbol, to be used as the return code from this EXEC. If the return code is not specified, it defaults to 0.

Usage Notes

1. If contrcl is returned to CMS, the CMS ready message indicates the return code value. Thus, the statement:

   ```
   &EXIT 12
   ```

   results in the ready message:

   ```
   R(00012);T=0/02 15:32:34
   ```

2. If you specify:

   ```
   &EXIT &RETCODE
   ```

   the return code value displayed is the return code from the most recently executed CMS command.

&GOTO

Use the &GOTO control statement to transfer control to a specific location in the EXEC procedure. Execution then continues at the location that is branched to. The format of the &GOTO control statement is:

```
┌──────────────────────────────────────────────────────────────────┐
│  &GOTO  │ ⎧ TOP         ⎫                                        │
│         │ ⎨ line-number ⎬                                        │
│         │ ⎩ -label      ⎭                                        │
└──────────────────────────────────────────────────────────────────┘
```

where:

TOP transfers control to the first line of the EXEC file.

line-number transfers control to a specific line in the EXEC file.

-label transfers control to a specific label in the EXEC file. A label must begin with dash (-), and it must be the first token on a line. The remainder of the line may contain an executable statement or it may be null.

Usage Notes

1. Scanning for an EXEC label starts on the line following the &GOTO statement, goes to the end of the file, then to the top of the file, and (if unsuccessful) ends on the line above the &GOTO statement. If more than one statement in the file has the same label, the first one encountered by these rules satisfies the search.

2. To provide a branch up or down a specific number of lines in the EXEC, use the &SKIP control statement.

&HEX

Use the &HEX control statement to initiate or inhibit hexadecimal conversion in an EXEC procedure. The format of the &HEX control statement is:

```
┌──────────────────────────────────────────────────────────────────┐
│  &HEX  │ ⎧ ON  ⎫                                                 │
│        │ ⎩ OFF ⎭                                                 │
└──────────────────────────────────────────────────────────────────┘
```

where:

ON indicates that tokens beginning with the string X' are to be interpreted as hexadecimal notation.

OFF indicates that no hexadecimal conversion is to be done by EXEC. OFF is the default setting.

Usage Notes

1. You should use the &HEX control statement when you want to display a hexadecimal value. For example:

   ```
   &HEX ON
   &TYPE X'40
   &HEX
   ```

   results in the display:

   ```
   28
   ```

   If you did not use the &HEX ON control statement, the &TYFE statement would result in the display:

   ```
   X'40
   ```

2. To convert a hexadecimal value to its decimal equivalent, use an assignment statement.

3. The *VM/370 CMS User's Guide* should be consulted for details and examples of correct usage of EXEC control statements with &HEX CN in effect.

&IF

Use the &IF control statement to test a condition in an EXEC procedure and to perform a particular action if the test is valid. If the test is invalid, execution continues with the statement following the &IF control statement. The format of the &IF statement is:

```
| &IF | ⎧token1⎫  operator  ⎧token2⎫  executable-statement |
|     | ⎨&$    ⎬            ⎨&$    ⎬                        |
|     | ⎩&*    ⎭            ⎩&*    ⎭                        |
```

where:

token1
token2 may be numeric constants, character strings, or EXEC variable symbcls. All variable symbols are substituted before the &IF statement is executed.

&$ tests all of the arguments entered when the EXEC was invoked. If at least one of the arguments satisfies the specified ccndition, the &IF statement is true.

&* tests all of the arguments entered when the EXEC was invoked. All of the entered arguments must meet the specified conditicn in order for the &IF statement to be true.

operator indicates the test to be performed on the tokens. If both tokens are numeric, an arithmetic test is performed. Otherwise, a logical (alphabetic) test is performed. The comparison operators, listed below, may be specified either in symbolic or mnemonic form:

Symbol	Operation
= or EQ	equals
¬= or NE	not equal
< or LT	less than
<= or LE	less than or equal to (not greater than)
> or GT	greater than
>= or GE	greater than or equal to (not less than)

executable-statement
: is any valid EXEC executable statement which may be a CMS command, an EXEC control statement, or an assignment statement. You may also specify another &IF statement; the number of &IF statements that may be nested is limited only by the record length of the file. In fixed-length EXEC files, only the first 72 characters of the line are scanned.

<u>Usage</u> <u>Notes</u>

1. The values &* and &$ are reset when an &ARGS or &READ ARGS contrcl statement is executed. They are not changed when you reset a specific numeric variable (&1 through &30).

2. If a variable symbol used in an &IF control statement is undefined, the EXEC interpreter cannot properly compare it. In cases where a variable may be null, or to check for a null symbol, you should use a concatenation character when you write the &IF statement; for example:

   ```
   &IF .&1 EQ . &GOTO -NOARGS
   ```

   tests for a null symbol &1.

3. If the symbols &* or &$ are null because no arguments were entered, the entire &IF statement is treated as a null statement.

&LOOP

Use the &LOOP control statement to describe a loop in an EXEC procedure, including the conditions for exit from the loop. The format of the &LOOP control statement is:

```
┌──────────────────────────────────────────────────────────────┐
│  &LOOP   │  ┌n     ┐┌m        ┐                                │
│          │  └-label┘└condition┘                                │
└──────────────────────────────────────────────────────────────┘
```

<u>where</u>:

n
: is a positive integer from 0 to 4C95 that indicates the number of executable and nonexecutable lines in the loop. These lines must immediately follow the &LOOP statement.

-label
: specifies that all of the lines following the &LCCP statement down to, and including the line with the specified label, are to be executed in the loop. The first character of the label must be a hyphen, and it must be the first token on a line. The remainder of the line may contain an executable statement, or it may be null.

m
: is a positive integer from 0 to 4C95 that indicates the number of times the loop is to be executed.

condition
: specifies the condition that must be met. The syntax of the exit condition is the same as that in the &IF statement, that is:

```
┌tok1┐ operator ┌tok2┐
│&$  │          │ &$ │
└&*  ┘          └ &* ┘
```

Usage Notes

1. When loop execution is complete, control passes to the next statement following the end of the loop.

2. The condition is always tested before the loop is executed. If the specified condition is met, then the loop is not executed. For example, the statement:

   ```
   &LOOP 3 &COUNT = 100
   ```

   specifies that the next three lines are interpreted until the value of &COUNT is 100.

3. Loops may be nested up to four levels deep. All nested loops may end at the same label.

&PUNCH

Use the &PUNCH control statement to punch a line of tokens to the virtual card punch. The format of the &PUNCH control statement is:

&PUNCH	[tok1 [tok2 ... [tokn]]]

where:

tok1 [tok2 ... [tokn]]
specifies the tokens to be punched. All tokens are padded or truncated to eight characters. The punched line is right-padded with blanks to fill an 80-column card. If no tokens are specified, a line consisting of 80 blank characters is punched.

Usage Notes

1. Lines punched with the &PUNCH control statement are scanned by the EXEC interpreter and variable symbols are substituted before the line is punched. In fixed-length EXEC files, only the first 72 characters of the record are scanned. To punch one or more lines of unscanned data, use the &BEGPUNCH or &BEGPUNCH ALL control statement.

2. When you have finished punching lines in an EXEC procedure, you can use the CP command CLOSE to close the spool punch file and release it for processing.

&READ

Use the &READ control statement to read one or more lines from the terminal or console stack. The lines may contain data or executable statements. The format of the &READ control statement is:

```
┌──────────────────────────────────────────────────────────────────┐
│         │ ┌                                   ┐                  │
│  &READ  │ │ n                                 │                  │
│         │ │ 1                                 │                  │
│         │ │ARGS                               │                  │
│         │ │VARS [&var1 [&var2 ... [&varn]]]   │                  │
│         │ └                                   ┘                  │
└──────────────────────────────────────────────────────────────────┘
```

where:

n — reads the next n lines from the terminal and treats them as if they had been in the EXEC file. Reading from the terminal stops when n lines have been read, or when an &LOOP statement or a statement that transfers control is encountered. If an &READ statement is encountered, the number of lines to be read by it is added to the number outstanding.

1 — If n is not specified, the default 1 is assumed, and the EXEC continues processing after reading a single line.

ARGS — reads a single line, assigns the entered tokens to the special variables &1, &2, ..., &n, and resets the special variables &INDEX, &*, and &$.

If any of the tokens is specified as a percent sign (%) or begins with the character X'FF', the corresponding argument is set to blanks.

VARS [&var1 [&var2 ... [&varn]]]
reads a single line and assigns the tokens entered to the variable symbols &var1, &var2, ..., &varn (up to 17).

These variables are scanned in the same way as though they appeared on the left-hand side of an assignment statement. If no variable names are specified, any data read from the terminal is lost.

If any of the tokens is specified as a percent sign (%) or begins with the character X'FF', the corresponding variable is set to blanks.

Usage Note

You can test the special variable &READFLAG to determine whether the next &READ statement will result in a physical read to your terminal (the value of &READFLAG is CONSOLE) or in reading a line from the console stack (the value of &READFLAG is STACK).

&SKIP

Use the &SKIP control statement to cause a specified number of lines in the EXEC file to be skipped. The format of the &SKIP control statement is:

```
┌──────────────────────────────────────────────────────────────────┐
│         │ ┌   ┐                                                  │
│  &SKIP  │ │ n │                                                  │
│         │ │ 1 │                                                  │
│         │ └   ┘                                                  │
└──────────────────────────────────────────────────────────────────┘
```

where:

n specifies the number of lines to be skipped:

- If n is greater than 0, the specified number of lines are skipped. Execution continues on the line following the skipped lines. If the value of n surpasses the number of lines remaining in the file, the EXEC terminates processing.
- If n is equal to 0, no lines are skipped, and execution continues with the next line.
- If n is less than 0, execution continues with the line that is n lines above the current line. An attempt to skip beyond the beginning of the file results in an error exit from the EXEC.
- The n may be coded as a variable symbol. 1 is the default value that is used when no value is specified for n.

Usage Note

To pass control to a particular label in an EXEC procedure, use the &GOTO control statement. The &GOTO control statement provides more flexibility when you want to update your EXEC procedures. The &SKIP statement, however, is more efficient, in terms of execution time.

&SPACE

Use the &SPACE control statement to display a specified number of blank lines at your terminal. The format of the &SPACE control statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
│            │  ┌   ┐                                                  │
│   &SPACE   │  │ n │                                                  │
│            │  │ 1 │                                                  │
│            │  └ ─ ┘                                                  │
└──────────────────────────────────────────────────────────────────────┘
```

where:

n specifies the number of blank lines to be displayed at the terminal. If no number is specified, &SPACE 1 is assumed by default.

Usage Notes

1. You may want to use the &SPACE control statement to control the format of the execution summary that displays while your EXEC executes.

&STACK

Use the &STACK control statement to stack a single data line in the console input stack. Stacked lines may be read by the EXEC, by CMS, or by the CMS editor. The format of the &STACK control statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
│            │ ┌    ┐ ┌                       ┐                        │
│  &STACK    │ │FIFO│ │ tok1 [tok2 ... [tokn ]]│                        │
│            │ │LIFO│ │ HT                    │                        │
│            │ └    ┘ │ RT                    │                        │
│            │        └                       ┘                        │
└──────────────────────────────────────────────────────────────────────┘
```

where:

FIFO specifies that the line is to be stacked in a first in, first out sequence, and is the default if not specified otherwise.

LIFO specifies that the line is to be stacked in a last in, first out sequence.

tok1 [tok2 ... [tokn]]
specify the tokens to be stacked. If no tokens are specified, a null line is stacked. The tokens are in expanded form.

HT stacks the CMS Immediate command HT (halt typing), which is executed immediately. All terminal display from the EXEC is suppressed until the end of the file or until an RT (resume typing) command is read.

RT stacks the CMS Immediate command RT (resume typing), which is executed immediately. If terminal display has been suppressed as the result of an HT (halt typing) request, display is resumed.

Usage Notes

1. Lines stacked with the &STACK control statement are scanned by the EXEC interpreter and variable symbols are substituted before the line is stacked. To stack one or more unscanned lines, use the &BEGSTACK or &BEGSTACK ALL control statement.

2. You must use the &STACK control statement when you want to stack a null line.

3. Any CMS Immediate command may be executed in an EXEC, using the &STACK control statement.

4. A complete discussion of techniques you can use to stack commands and data in the console stack is provided in the *VM/370 CMS User's Guide*.

&TIME

Use the &TIME control statement to request timing information to be displayed at the terminal after each CMS command that is executed. The format of the &TIME control statement is:

```
┌──────────────────────────────────────────┐
│          │ ┌     ┐                       │
│  &TIME   │ │ON   │                       │
│          │ │OFF  │                       │
│          │ │RESET│                       │
│          │ │TYPE │                       │
│          │ └     ┘                       │
└──────────────────────────────────────────┘
```

where:

ON resets the processor's time before every CMS command, and prints the timing information on return. If the &CONTROL control statement is set to CMS or ALL, the display of the timing information is followed by a blank line.

OFF does not automatically reset the processor's time before every CMS command, nor does it print the timing information on return.

RESET performs an immediate reset of the processor's time.

TYPE displays the current timing information (and resets the processor's time).

Usage Notes

1. When timing information is displayed, it is in the format:

   ```
   T=x.xx/y.yy hh:mm:ss
   ```

   where:

   x.xx is the virtual processor's time used since it was last reset in the current EXEC file.

   y.yy is the total of the processor's time used since it was last reset in the current EXEC file.

   hh:mm:ss is the actual time of day in hours:minutes:seconds.

2. The processor's time is set to zero before the execution of the first statement in the EXEC file, and is again set to zero (reset) whenever timing information is printed.

&TYPE

Use the &TYPE control statement to display a line of tokens at the terminal. The format of the &TYPE control statement is:

```
| &TYPE  | [tok1 [tok2 ... [tokn]]]                      |
```

<u>where</u>:

tok1 [tok2... [tokn]]
specify the tokens to be displayed. All tokens are padded or truncated to eight characters. If no tokens are specified, a null line is displayed.

<u>Usage Note</u>

Lines displayed with the &TYPE control statement are scanned by the EXEC interpreter and variable symbols are substituted before the line is displayed. To display one or more unscanned lines, use the &BEGTYPE or &BEGTYPE ALL control statements.

Built-In Functions

You can use the EXEC built-in functions to assign and manipulate variable symbcls. With the exception of &LITERAL, built-in functions may be used only on the right-hand side of an assignment statement, as follows:

```
&MIX = &CONCAT &1 &2
```

Built-in functions may not be combined with arithmetic expressions.

Each of the built-in functions (&CONCAT, &DATATYPE, &LENGTH, &LITERAL, and &SUBSTR) is described separately.

&CONCAT

Use the &CONCAT function to concatenate two or more tokens and assign the result to a variable symbol. The format of the &CONCAT function is:

```
&variable = &CONCAT tok1 [tok2 ... [tokn]]
```

where:

&variable is the variable symbol whose value is determined by the &CONCAT function.

tok1 [tok2...[tokn]]
specifies the tokens that are to be concatenated into a single token; for example:

```
&A = **
    .
    .
    .
&B = &CONCAT XX &A 45
&TYPE &B
```

results in the printed line:

```
XX**45
```

Usage Note

If the concatenated token is longer than eight characters, the data is left-justified and truncated on the right.

&DATATYPE

Use the &DATATYPE function to determine whether the value of the specified token is alphabetic or numeric data. The format of the &DATATYPE function is:

```
| &variable = &DATATYPE  token                                       |
```

where:

&variable is the variable symbol whose value is determined by the &DATATYPE function.

token specifies the target token that is to be examined for alphabetic or numeric data. The result of the &DATATYPE function has the value NUM or CHAR, depending on the data type of the specified token. For example:

```
&CHECK = &DATATYPE  ABC
&TYPE &CHECK
```

results in the display:

```
CHAR
```

A null token is considered character data.

&LENGTH

Use the &LENGTH function to determine the number of characters in a token. The format of the &LENGTH function is:

```
| &variable = &LENGTH  token                                         |
```

where:

&variable is the variable symbol whose value is determined by the &LENGTH function.

token specifies the target token that is to be examined for nonblank characters. The result of the &LENGTH function is the number of nonblank characters in the specified token. For example:

```
&LEN = &LENGTH ALPHA
&TYPE &LEN
```

results in the display:

```
5
```

&LITERAL

Use the &LITERAL function to inhibit variable substitution on the specified token. The &LITERAL function may appear in any EXEC contrcl statement, as follows:

```
[...] &LITERAL token[...]
```

where:

token specifies the token whose literal value is to be used without substitution. For example:

```
&X = **
&TYPE &LITERAL &X EQUALS &X
```

results in the printed line:

```
&X EQUALS **
```

&SUBSTR

Use the &SUBSTR function to extract a character string from a specified token and tc assign the substring to a variable symbol. The format cf the &SUBSTR function is:

```
&variable = &SUBSTR  token i [j]
```

where:

&variable is the variable symbol whose value is determined by the &SUBSTR function.

token is the token from which the character string is to be extracted.

i specifies the character position in the token of the first character to be used in the substring.

j specifies the number of characters in the string. If omitted, the remainder of the token is used.

Usage Note

The values of i and j (if given) must be positive integers. For example:

```
&A = &SUBSTR ABCDE 2 3
&TYPE &A
```

results in the printed line:

```
BCD
```

Special Variables

Special variables are variable symbols that are assigned values by the EXEC interpreter, and that you can test or display in your EXEC procedures. In some cases, you may assign your own values to EXEC special variables; these cases are noted in the variable descriptions.

&n

The &n special variable represents the numeric variables &1 through &30. When an EXEC is invoked, the numeric variables from &1 through &30 are initialized according to the arguments that are passed to the EXEC file (if any).

The numeric variables can be reset by either an &ARGS or &READ ARGS control statement; when fewer than 30 arguments are set or reset, the remainder of the &n variables are set to blanks. A particular argument can be set to blanks by assigning it a percent sign (%) when invoking the EXEC procedure, in an &ARGS control statement, or in an &READ ARGS control statement. An argument is also set to blanks if it begins with the character X'FF' and is specified when invoking the EXEC procedure or in an &READ ARGS control statement.

You may set the values of specific arguments using assignment statements. Any value of n, however, that is greater than 30 or less than 0 is rejected by the EXEC interpreter.

&* and &$

These variables can be used to perform a collective test on all of the arguments passed to the EXEC procedure. &* and &$ may only be used in the &IF and &LOOP control statements and are described under the description of the &IF control statement.

You may not assign values to the special variables &* and &$.

&O

The &0 special variable contains the filename of the EXEC file. You may test and manipulate this variable.

&DISKx

You can use the &DISKx special variable to determine whether a disk is an OS, DOS, or CMS disk. x represents the mode letter at which the disk is accessed. For example, if you access an OS disk with a mode letter of C, then the special variable &DISKC has a value of OS. The possible values for the &DISKx special variable are OS (for an OS disk), DOS (for a DOS disk), CMS (for a CMS disk), and NA (when the disk is not accessed).

You may set or change the values of an &DISKx special variable; if you do so, however, you will no longer be able to test the status of the disk at mode x.

&DISK*

The &DISK* special variable contains the one-character mode letter cf the first read/write disk in the CMS search order. If you have no read/write disks accessed, this special variable contains the value NONE.

You may assign a value to the &DISK* special variable for your own use; if you do so, however, you will not be able to use it to obtain the filemode letter of a read/write disk.

&DISK?

You can use the &DISK? special variable in an EXEC to determine which read/write disk that you have accessed has the most space on it. If you have no read/write disks accessed, &DISK? contains the value NONE.

You may assign a value to the &DISK? special variable for your own use; if you do so, however, you will no longer be able to locate the read/write disk with the most space.

&DOS

The &DOS special variable contains one of the two character values ON or OFF, depending on whether the CMS/DOS environment is active. If you have issued the command:

```
     set dos on
```

then the &DOS special variable contains the value ON.

You may set or change the value of the &DOS special variable for your own use; if you do so, however, you will not be able to test whether the CMS/DOS environment is active.

&EXEC

The &EXEC special variable is the filename of the EXEC file. You cannot set this variable explicitly but you can examine and test it.

&GLOBAL

The &GLOBAL special variable contains the recursion level of the EXEC currently executing. Since the EXEC interpreter can handle up to 19 levels of recursion, the value of &GLOBAL ranges from 1 to 19. You cannot set this variable explicitly, but you can examine and test it.

&GLOBALn

The &GLOBALn special variable represents the variables &GLOBAL0 through &GLOBAL9. You can set these variables only to integral numeric values. They are all initially set to 1. Unlike other EXEC variables, these can be used to communicate between different recursion levels of the EXEC interpreter.

&INDEX

The &INDEX special variable contains the number of arguments passed to the EXEC procedure. Since up to 30 arguments can be passed to an EXEC procedure, the value of &INDEX can range from 0 through 30.

Although you cannot set this variable explicitly, it is reset by an &ARGS or &READ ARGS control statement. &INDEX can be examined to determine the number of active arguments in the EXEC procedure.

&LINENUM

The &LINENUM special variable contains the current line number in the EXEC file. You cannot explicitly set this variable but you can examine and test it.

&READFLAG

The &READFLAG special variable contains one of two literal values: CONSOLE or STACK. If there are stacked lines in the terminal input buffer (console stack) &READFLAG contains the value STACK and the next read request results in a line being read from the stack. If not, then the next read request results in a physical read to the terminal, and the value of &READFLAG is CONSOLE. You cannot explicitly set this variable but you can examine and test it.

&RETCODE

The &RETCODE special variable contains the return code from the most recently executed CMS command. &RETCODE can contain only integral numeric values (positive or negative), and is set after each CMS command is executed. You can examine, test, and change this variable but changing it is not recommended.

&TYPEFLAG

The &TYPEFLAG special variable contains one of two literal values: RT (resume typing) or HT (halt typing). It contains the value HT when terminal display has been suppressed by the Immediate command HT. It contains the value RT when the terminal is displaying output. You cannot explicitly set this variable, but you can examine and test it.

Section 6. CMS Macro Instructions

This section describes the formats of the CMS assembler language macros, which you can use 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 CMSLIB MACLIB, which is the macro library (located on the system disk) which contains CMS macros. To assemble a program to execute in a CMS environment that includes VM/370 Basic System Extensions (Program No. 5748-XX8), you must add DMSB20 MACLIB (also on the system disk) to the GLOBAL command statement.

For functional descriptions and usage examples of the CMS macros, see the VM/370 CMS User's Guide.

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.

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 or TXTLIB; it is loaded via the CMS INCLUDE command.

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

2. 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.

3. 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.

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. 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.

2. The FSCBD macro instruction expands as follows:

```
          FSCBD
FSCBD     DSECT
FSCBCOMM  DS    CL8        Command
FSCBFN    DS    CL8        Filename
FSCBFT    DS    CL8        Filetype
FSCBFM    DS    CL2        Filemode
FSCBITNO  DS    H          Relative record (item) number
FSCBBUFF  DS    A          Address of read/write buffer
FSCBSIZE  DS    F          Length of buffer
FSCBFV    DS    CL1        Record format (F or V)
FSCBFLG   DS    X          PLIST flag
FSCBNOIT  DS    H          Number of records to be read/written
FSCBNORD  DS    A          Number of records actually read
FSCBAITN  DS    F          Extended item number
```

```
FSCBANIT  DS    F           Extended number cf items
FSCBWPTR  DS    F           Write pointer
FSCBRPTR  DS    F           Read pointer
```

3. If you specify FORM=E as the parameter of the FSCB marco 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 and save its current status on disk. The format of the FSCLOSE macro instruction is:

```
┌──────────────────────────────────────────────────────────────────────┐
│ [label] │ FSCLOSE │ { fileid[,FSCB=fscb] } [,ERROR=erraddr]          │
│         │         │ { FSCB=fscb          }                           │
└──────────────────────────────────────────────────────────────────────┘
```

__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.

Error Conditions

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

Code	Meaning
6	File not open

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] } [,ERROR=erraddr]     |
|         |         | { FSCB=fscb          }                      |
```

__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, third, and fourth words of the list contain the filename, filetype, and 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] } [,ERROR=erraddr][,options]|
|         |        | { FSCB=fscb           }                           |
└──────────────────────────────────────────────────────────────────────┘
```

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.

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 to create an FSCB for it.

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         }
|         |         | [,WPTR=wptr]      [,RPTR=rptr] [,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 you don't code 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.

WPTR=wptr specifies the new value of the write pointer.

number   any assembler symbol or number.
(reg)   a register containing the binary number.

RPTR=rptr specifies the new value of the read pointer.

number   any assembler symbol or number.
(reg)   a register containing the binary number.

FORM=E specifies 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 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
20	Invalid character in fileid
24	Invalid filemode
28	File not found
36	Disk not accessed

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

| specifies 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.

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.

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
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 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
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
| specifies that the extended format FSCB is being used.

Usage Notes

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

2. When the FSSTATE macro completes execution, register 1 contains the address of the file status table (FST) for the specified file.

   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)
32	Logical record length
36	Number of 800-byte data blocks
38	Year (yy) last written

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
| specifies that 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 and a number. 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 blocksize 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.

Error Conditions

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

Code	Meaning
2	Invalid buffer address
4	First character of filemode is invalid
5	Second character of filemode is invalid
6	Item number too large (more than 65,533 for an 800-byte formatted disk)
7	Attempt to skip over unwritten variable-length item
8	Buffer size not specified
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

Code	Meaning
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 65K 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.

HNDEXT

Use the HNDEXT macro instruction to trap external interruptions and pass control to an internal routine for processing. External interruptions are caused, in a virtual machine, 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

Register 13 points to the user save area at label UAREA.

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

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 scon as the interruption occurs.

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

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

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

Usage Notes

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

2. If you specify WAIT, then 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, then the routine at the specified address receives control immediately.

3. 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 0 in register 15 means that you are through handling the interruption; any nonzero return code indicates that you expect another interruption.

4. 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.

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 contrcl if an error is found. If ERROR= is not coded and an errcr 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 fcr your SVC-handling routine. When your program receives control, the register contents are as follows:

Register	Contents
12	Address of your SVC-handling routine
13	Address of an 18-fullword save area (for your use)
14	Return address

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 or address
2	SVC number set replaced previously set number
3	SVC number cleared was not set

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:

```
┌──────────────────────────────────────────────────────────────────────────┐
|         |          | ┌                    ┐ ┌           ┐┌              ┐|
| [label] | LINEDIT  | |,TEXT='messagetext' | |,DOT=(YES)||  ,COMP=(YES)||
|         |          | |,TEXTA=address      | |     (NO )||         (NO )||
|         |          | └                    ┘ └           ┘└              ┘|
|         |          | [,SUB=(substitutionlist)]                          |
|         |          | ┌                   ┐ ┌                     ┐       |
|         |          | |,DISP= (TYPE  )    | |,BUFFA=((address))   |       |
|         |          | |       (NONE  )    | |        ( (reg)  )   |       |
|         |          | |       (SIO   )    | └                     ┘       |
|         |          | |       (PRINT )    |                               |
|         |          | |       (CPCOMM)    |                               |
|         |          | |       (ERRMSG)    |                               |
|         |          | └                   ┘                               |
|         |          | ┌                          ┐ [,MAXSUBS=number]      |
|         |          | |,MF= (I                 ) |                        |
|         |          | |     (L                 ) |                        |
|         |          | |     (((E,address))     ) |                        |
|         |          | |     ( ((reg)     )     ) |                        |
|         |          | └                          ┘                        |
|         |          | ┌              ┐                                    |
|         |          | |,RENT= (YES)  |                                    |
|         |          | |       (NO )  |                                    |
|         |          | └              ┘                                    |
└──────────────────────────────────────────────────────────────────────────┘
```

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.

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 nct 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 cr 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 cf 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, ycu can use the list and execute forms of the macro to write reentrant code (see "MF Operand").

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:

$$\text{TEXTA=}\begin{Bmatrix}\text{label}\\\text{(reg)}\end{Bmatrix}$$

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 forms 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:

$$\text{DOT=}\begin{Bmatrix}\underline{\text{YES}}\\\text{NO}\end{Bmatrix}$$

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.
```

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}                           |
       < DECA {,(reg)  }                           >
       |                                           |
       | HEX4A  {,address                      }   |
       | CHARA  {,(reg)                        }   |
       | CHAR8A {,({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 nct 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. Thus to display 8 bytes of information (16 hexadecimal digits), you must code 17 periods in the message text.

For example, 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 form 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 forms 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 opticn 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 as such does not appear on the printed output. (See the discussion of the PRINTL macro for a list cf valid ASA control characters.)

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

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

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

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

The standard format of VM/370 error messages is:

```
xxxmmmnnns
```

where xxxmmm is the name of the module issuing the message, nnn is the message number, and 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/370 message types:

Code	Message Type
I	Information
W	Warning
E	Error

Then, the line is displayed in accordance with the CP EMSG setting. If EMSG is set to ON, then the entire message is displayed; if EMSG is set tc TEXT, then only the message portion is displayed; if EMSG is set to CODE, then 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={address}
      { (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 cf the text is inserted in subsequent bytes.

If you use register notation to indicate the tuffer 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 forms when you write reentrant programs. The format cf the MF operand is:

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

where:

MF=I (Standard form)
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 form)
generates a parameter list to be filled in when the execute form cf the macrc is used.

The size of the area reserved depends upon the number cf 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,address) (Execute form)
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 form of the macro. The following example shows how you might use the list and execute forms 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 form of the LINEDIT macro instruction is used, the parameter list for the message is built at label LINELIST, where the list form of the macro was coded.

MAXSUBS Operand

Use the MAXSUBS operand when you code the list form (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 form of the macro is used.

RENT Operand

Use the RENT operand when you are going to use the standard form 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 form of the macro instruction and you specify one of the following:

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

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.

PRINTL

Use the PRINTL macro instruction to write a line to a virtual printer. The format of the PRINTL macro instruction is:

[label]	PRINTL	line [,length] [,ERROR=erraddr]

where:

label   is an optional statement label.

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 containing the address of the line.

length   specifies the length of the line to be printed. (See Note 1.) It may be:

(reg)	a register containing the length.
n	a self-defining term indicating the length.

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. The maximum length allowed is 151 characters on a virtual 3211 or 133 characters on a virtual 1403 or 3203. 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.

2. 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
b̸	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

3. 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.

4. If the line does not begin with a valid carriage control character, the line is printed with a write command to space one line before printing (ASA X'40').

5. When the macro completes, register 15 may contain a 2 or a 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 to print particular information at the bottom or at the end of each page being printed.

   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).

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.

Error Conditicns

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

Code	Meaning
1	Line too long
2	Channel 12 punch sensed (virtual 3203 or 3211 only)
3	Channel 9 punch sensed (virtual 3203 or 3211 only)
4	Intervention required
5	Unknown error
100	Printer not attached

PUNCHC

Use the PUNCHC macro instruction to write a line to a virtual card punch. The fcrmat 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 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

RDCARD

Use the RDCARD macro instruction to read a line from a virtual card reader. The format of the RDCARD macro instruction is:

[label]	RDCARD	buffer[,length][,ERROR=erraddr]

where:

label — is an optional statement label.

buffer — specifies the buffer address into which the card is to be read. It may be:

bufaddr — the symbolic address of the buffer.
(reg) — a register containing the address of the buffer.

length — specifies the length of card to be read. If omitted, 80 is assumed. The length may be specified in one of two ways:

n — a self-defining term indicating the length.
(reg) — a register containing the length.

ERROR=erraddr
specifies the address of an error routine to be given ccntrol 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.

Error Conditicns

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 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 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:

```
TAPn      indicates the symbolic tape number (TAP1 through
          TAP4).
cuu       indicates the virtual device address.
```

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=0).
       9  indicates a 9-track tape (implies density=800).

density   200, 556, or 800 for a 7-track tape.
          800, 1600, or 6250 for a 9-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. 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.

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
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]|,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 containing the address of the buffer.

EDIT=code specifies the type of editing, if any, to be performed on the input line.

NO 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 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.

Error Conditions

When 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	
R0	EQU	0
R1	EQU	1
R2	EQU	2
R3	EQU	3
R4	EQU	4
R5	EQU	5
R6	EQU	6
R7	EQU	7
R8	EQU	8
R9	EQU	9
R10	EQU	10
R11	EQU	11
R12	EQU	12
R13	EQU	13
R14	EQU	14
R15	EQU	15

Extended	Control	Registers
C0	EQU	0
C1	EQU	1
C2	EQU	2
C3	EQU	3
C4	EQU	4
C5	EQU	5
C6	EQU	6
C7	EQU	7
C8	EQU	8
C9	EQU	9
C10	EQU	10
C11	EQU	11
C12	EQU	12
C13	EQU	13
C14	EQU	14
C15	EQU	15

Floating-Point	Registers	
F0	EQU	0
F2	EQU	2
F4	EQU	4
F6	EQU	6

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]

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 gap
BSR	Backspace one record
BSF	Backspace one file
FSR	Forward-space one record
FSF	Forward-space one file
WTM	Write a tape mark

device specifies the tape on which the control operation is to be performed. If omitted, TAP1 (virtual address 181) is assumed. It may be:

TAPn indicates the symbolic tape number (TAP1 through

TAP4).

cuu indicates the virtual device address.

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=0).

9 indicates a 9-track tape (implies density=800).

density 200, 556, or 800 for a 7-track tape.
800, 1600, or 6250 for a 9-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 Note

You need not specify the MODE option when you are manipulating a 9-track tape and you are using the default density for the tape drive, nor when you are writing a 7-track tape with a density of 800 bpi, odd parity, with data converter and translator off.

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

TAPESL

The TAPESL macro processes IBM standard HDR1 and EOF1 labels without using DOS or OS OPEN and CLOSE macros. This macro is used in conjunction 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]    |
|         |        | [,BLKCT=blkct][,ERROR=erraddr]                     |
|         |        | ┌            ┐┌          ┐                        |
|         |        | |,SPACE={YES}||,TM={YES}|                        |
|         |        | |       {NO } ||    {NO }|                        |
|         |        | └            ┘└          ┘                        |
└──────────────────────────────────────────────────────────────────────┘
```

where:

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:
TAPn n=1-4. If omitted, 181 is assumed.
cuu 181-184 are the only values allowed.

MODE=mode specifies the number of tracks, density, and tape recording technique options. It must be in this 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).

density 200, 556, or 800 for a 7-track tape.
800, 1600, or 6250 for a 9-track tape.

trtch indicates the tape recording technique for 7-track tape. One of these 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.)

BLKCT=blkct

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, HOUT, 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=$\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\}$

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=$\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\}$

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. 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 defaulted) 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.

2. 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 this publication for information about the default fields.

3. For a more complete discussion of tape label processing, see the section "CMS Tape Label Processing" in the *VM/370 CMS User's Guide*.

Error Conditions

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

Code	Meaning
24	Invalid device type specified.
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.

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 contrcl 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 Conditicns

When an error is detected, register 15 contains a 1 to indicate that an invalid device number was specified.

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

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]                                  |
```

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 time.

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:

TAPn   indicates the symbolic tape number (TAP1 through TAP4).
cuu   indicates the virtual device address.

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).

density 200, 556, or 800 for a 7-track tape
800, 1600, or 6250 for a 9-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 Note

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.

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

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 quotation marks.
lineaddr	the label on the statement containing the line.
(reg)	a register containing the address of the line.

length — specifies 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:

<u>YES</u> 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:

<u>B</u> 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.

<u>Usage Notes</u>

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.

Section 7. HELP Format Words

This section describes the formats, operands, and defaults of the HELP facility format words. In each of the format word descriptions, the default values are those that are implied when you enter a format word with no operands or parameters. For example, the default operand of the .FO (FORMAT MODE) format word is 'on'. Therefore, the format lines

```
     .fo
     .fo on
```

are equivalent, and in the format box of the .FO format word the 'on' operand is underscored.

HELP format words are used only in HELP description files when the user wants HELP to do output formatting when the file is processed. Figure 20.1 is a summary of the HELP facility format words.

Format word	Operand Format	Function	Break	Default Value
.BX (BOX)	V1 V2 ...Vn OFF	Draws horizontal and vertical lines around subsequent output text in blank columns.	Yes	Draws a horizontal line.
.CM (COMMENT)	Comments	Places comments in a file for future reference.	No	
.CS (CONDITIONAL SECTION	n ON/OFF	Allows conditional inclusion of input in the formatted output.	No	
.FO (FORMAT -MODE)	ON/OFF	Causes concatenation of input lines, and left and right-justification of output.	Yes	On
.IL (INDENT LINE)	n|+n|-n	Indents only the next line the specified number of spaces.	Yes	0
.IN (INDENT)	n|+n|-n	Specifies the number of spaces subsequent text is to be indented.	Yes	0
.OF (OFFSET)	n|+n|-n	Provides a technique for indenting all but the first line of a section.	Yes	0
.SP (SPACE)	n	Specifies the number of blank lines to be inserted before the next output line.	Yes	1
.TR(TRANSLATE)	s t	Specifies the final output representation. of any input character.	No	

Figure 20.1. HELP Format Word Summary

.BX (BOX)

The BOX format word defines and initializes a horizontal rule for output and defines vertical rules for subsequent output lines.

The format of the .BX format word is:

```
┌──────────────────────────────────────────────────────────────────────┐
│              │             ┌               ┐                          │
│    .BX       │             │v1 v2 [...[vn]]│                          │
│              │             │OFF            │                          │
│              │             └               ┘                          │
└──────────────────────────────────────────────────────────────────────┘
```

where:

vl-vn are the positions at which you want to plae vertical rules in output text. This format of the format word initializes the box and draws a horizontal line with vertical descenders at the columns indicated. Subsequently entering the .BX format word with no operands causes HELP to print a horizontal line with vertical bars at the columns indicated.

Off causes HELP to finish drawing the box by printing a horizontal line with vertical ascenders at the columns specified in a previous .BX format word.

Usage Notes

1. The .BX format word describes an overlay structure for subsequent text that is processed by HELP. After the '.BX v1 v2 ...' line is processed, HELP continues processing output lines as usual. However, before a line is printed, HELP places vertical bars in the columns indicated by v1, v2, and so on, unless a column is already occupied by a data character. In this case, HELP does not place a vertical bar in the column.

2. The .BX control word causes a break in the text.

3. The terminal output characters for boxes are formed with dashes (-), vertical bars (|), and plus signs (+).

4. You can specify a .BX format word with different columns while a box is being drawn. When this happens, HELP puts in vertical ascenders for all the old columns and vertical descenders for all the new columns. The vertical rules then appear in all subsequent output lines in the new columns designated.

5. The column specification for the .BX format word uses a different rule than is used elsewhere in HELP. In some control words the numbers in the format word represent not columns but displacements. For example the HELP format word .IN 5 means that a blank character should be expanded to enough blanks to fill up through column 5; the next word starts in column 6. In the .BX control word, .BX 5 means to put vertical rules in column 5. Thus, you can use the same numbers for a .IN control word as for a .BX control word, and the vertical bar will appear in the column immediately preceding the first word on that line.

Example

Consider the HELP file called 'MARYHADA' that looks like this:

```
     .bx 1 43
     .in 5
     Mary had a little lamb,
     Whose fleece was white as snow,
     And everywhere that Mary went,
     The lamb was sure to go.
     .bx off
```

This file, when processed by HELP, creates the following output:

```
┌─────────────────────────────────────────┐
│    Mary had a little lamb,              │
│    Whose fleece was white as snow,      │
│    And everywhere that Mary went,       │
│    The lamb was sure to go.             │
└─────────────────────────────────────────┘
```

.CM (COMMENT)

Use the COMMENT format word to place comments within a HELP file.

The format of the .CM format word is:

.CM	comments

where:

comments may be anything; this input line is not used in formatting the output.

Usage Notes

1. The .CM format word enables you to store comments in the HELP files for future reference. The comments can be seen only by editing the HELP file.

2. You can use comments to store unique identifications to be used to locate a specific region of the file during editing.

Example

```
    .CM Remember to change the date.
```

The line above is seen only when editing the HELP file, and it reminds you to change the date used in the text.

.CS (CONDITIONAL SECTION)

The CONDITIONAL SECTION format word identifies to HELP the sections of the input file that are to be conditionally processed based on the specified HELP command option.

The format of the .CS format word is:

```
┌─────────────┬──────────────────────────────────────────────────────┐
│             │                                                      │
│    .CS      │            n [ ON  ]                                 │
│             │              [ OFF ]                                 │
│             │                                                      │
└─────────────┴──────────────────────────────────────────────────────┘
```

where:

n   specifies the conditional section code number from 1 to 3.

on   marks the beginning of conditional section n.

off marks the end of conditional section n.

Usage Notes

1. The .CS format word enables you to identify the specific sections of the input file that are directly associated with the HELP facility command 'options', and that will be included in the output based on the HELP command option specified.

   If you choose to implement any HELP description files using the ALL, PARM, FORM, and DESC options, the format word .CS is required in the file. You must use the following form:

   ```
   Top of file
   .CS 1 on
       (Text for DESC option)
   .CS 1 off
   .CS 2 on
       (Text for FORM option)
   .CS 2 off
   .CS 3 on
       (Text for PARM option)
   .CS 3 off
   End of file
   ```

2. A conditional section can contain HELP format words as well as text. If the section is ignored when processed by HELP, all format words contained in that section are ignored, except the format word:

   ```
   .CS n off
   ```

   which marks the end of the section.

3. Imbedding .CS format words (that is, specifying the beginning of a conditional section before you have specified the end of a previous conditional section) produces unpredictable results.

.FO (FORMAT MODE)

Use the FORMAT MODE format word to cancel or restore concatenation of input lines and right-justification of output lines.

The format of the .FO format word is:

```
┌──────────────────────────────────────────────────────────────────────┐
│           │            ┌     ┐                                       │
│    .FO    │            │ ON  │                                       │
│           │            │ OFF │                                       │
│           │            └     ┘                                       │
└──────────────────────────────────────────────────────────────────────┘
```

where:

ON restores default HELP formatting, including both justification and concatenation of lines. If you use the .FO format word with no operands, ON is assumed.

Off cancels concatenation of input lines and justification of output lines. Subsequent text is printed 'as is'.

Usage Notes

1. When format mode is in effect, lines are formed by shifting words to or from the next line (concatenation) and padding with extra blanks to produce an aligned right margin (justification).

2. This format word acts as a break.

3. When format mode is in effect, a line without any blanks that exceeds the current line length is extended into the right margin. If a line is processed so that only one word fits on the line, the word is left-justified.

4. If no formatting is to be done by HELP, HELP description files must contain a '.fo off' format word as the first line of the file.

Examples

1. .FO off

```
Justification and concatenation are
completed for
the preceding line or lines, but the following
lines are
typed exactly as they appear in the file.
```

2. .FO

Justification and formatting are resumed with the next input line. Output from this point on in the file is padded to produce an aligned right margin on the output page.

.IL (IDENT LINE)

Use the INDENT LINE format word to indent the next line only a specified number of characters.

The format of the .IL format word is:

```
┌──────────────────────────────────────────────────────────────┐
│           │             ┌    ┐                               │
│    .IL    │             │ n  │                               │
│           │             │ +n │                               │
│           │             │ -n │                               │
│           │             └    ┘                               │
└──────────────────────────────────────────────────────────────┘
```

where:

n specifies the number of character spaces to shift the next line from the current margin. +n specifies that text is shifted to the right, and -n shifts text to the left.

Usage Notes

1. The .IL format word provides a way to indent the next output line. The line is shifted to the right or the left of the current margin (which includes any indent or offset values in effect).

2. This format word acts as a break.

3. The .IL format word is useful for beginning new paragraphs.

4. When successive .IL format words are encountered without intervening text, or when you specify positive or negative increments for .IL format words entered without intervening text, the indent amount is modified to reflect the last .IL encountered; that is, the increments are added together. Thus the lines:

   ```
   .il 4
   .il +6
   ```

   result in the next line being indented 10 spaces.

5. When you use the .IL format word with a negative value (undenting), an error message is generated if the resulting amount would cause a shift to the left of character position one.

.IN (INDENT)

Use the INDENT format word to change the left margin displacement of HELP output.

The format of the .IN format word is:

```
┌──────────────────────────────────────────────────────────────────┐
│             │          ┌    ┐                                    │
│    .IN      │          │ n  │                                    │
│             │          │ +n │                                    │
│             │          │ -n │                                    │
│             │          │ 0  │                                    │
│             │          └    ┘                                    │
└──────────────────────────────────────────────────────────────────┘
```

where:

n specifies the number of spaces to be indented. If omitted, 0 is assumed, and indentation reverts to the left margin. If you use +n or -n, the current left margin increases or decreases by the amount specified.

Usage Notes

1. The .IN format word resets the current left margin. This indentation remains in effect for all following lines until another .IN format word is encountered. '.IN 0' cancels the indentation, and output continues at the original left margin setting.

2. The value of n represents the number of blank spaces left before text margins. Thus, '.in 5' sets the left margin at column 6, leaving 5 blank spaces at the left.

3. This format word acts as a break.

4. The .IN format word cancels any .OF (OFFSET) setting. The .OF 0 request cancels the current offset, but leaves the left margin specified by the .IN format word unchanged.

Example

1. .in 10

   All lines processed after this request are indented 10 spaces from the current left margin setting. This indentation continues until another .IN format word is encountered.

2. .in 0

   The effect of any current indentation is canceled, and output continues at the original left margin setting.

.OF (OFFSET)

Use the OFFSET format word to indent all but the first line of a block of text.

The format of the .OF format word is:

```
┌─────────────────────────────────────────────────────────────────────┐
│            │           ┌    ┐                                       │
│    .OF     │           │ n  │                                       │
│            │           │ +n │                                       │
│            │           │ -n │                                       │
│            │           │ 0  │                                       │
│            │           └    ┘                                       │
└─────────────────────────────────────────────────────────────────────┘
```

where:

n specifies the number of spaces to be indented after the next line is formatted. If omitted, 0 is assumed, and indentation reverts to the original margin setting. If you use +n or -n, the current offset value increases or decreases the specified amount, and a new offset is started.

Usage Notes

1. The .OF format word does not take effect until after the next line is formatted. The indentation remains in effect until a .IN (INDENT) format word or another OFFSET control word is encountered.

   You can use the .OF format word within a section that is also indented with the .IN format word. Note that .IN settings take precedence over .OF, however, and any .IN request causes a previous offset to be cleared.

   If you want to start a new section with the same offset as the previous section, you need only repeat the .OF n request.

2. This format word acts as a break.

3. You can use the .IL (INDENT LINE) format word to shift only the next line to the left or right of the current margin.

Example

1. Starting an offset:
   .of 10

   The line immediately following the .OF format word is printed at the current left margin. All lines thereafter (until the next indent or offset request) are indented ten spaces from the current margin setting. These two examples were processed with OFFSET control words in the positions shown.

2. Ending an offset:

   .of

The effect of any previous .OF request is canceled, and all output after the next line continues at the current left margin setting.

.SP (SPACE LINES)

Use the SPACE LINES format word when you want blank lines to appear between text lines of output.

The format of the .SP format word is:

```
┌──────────────────────────────────────────────────────────────────┐
│          │            ┌     ┐                                     │
│   .SP    │            │  n  │                                     │
│          │            │  1  │                                     │
│          │            └     ┘                                     │
└──────────────────────────────────────────────────────────────────┘
```

__where__:

n specifies the number of blank lines to be inserted in the output. If omitted, 1 is assumed.

.TR (TRANSLATE CHARACTER)

The TRANSLATE CHARACTER format word allows you to specify the output representation of each character in the source text. For example, you could specify that all exclamation points in the file appear as blanks in the output.

The format of the .TR format word is:

.TR	[s t]

where:

s — is a source character under consideration. It may be a single character or a two-character hexadecimal code.

t — is the intended output representation of the source character. It may be a single character or a two-character hexadecimal code.

Usage Notes

1. After formatting of an input source line has been completed and immediately before actual output, each character of the output line may be translated to a different output code.

2. Since format words are only processed internally, they are never translated in the file.

3. Translate character specifications remain in effect until explicitly respecified.

4. A .TR format word with no operands causes the translation table to be reinitialized and all previously specified translations to be reset.

5. The .TR format word does not cause a break. If you have a section of text that has translation characters in effect, followed by a .TR to reset the translations, the last line of the text may not yet have been printed. In this case, that last line is not translated.

Example

```
    .tr 40 ?
```

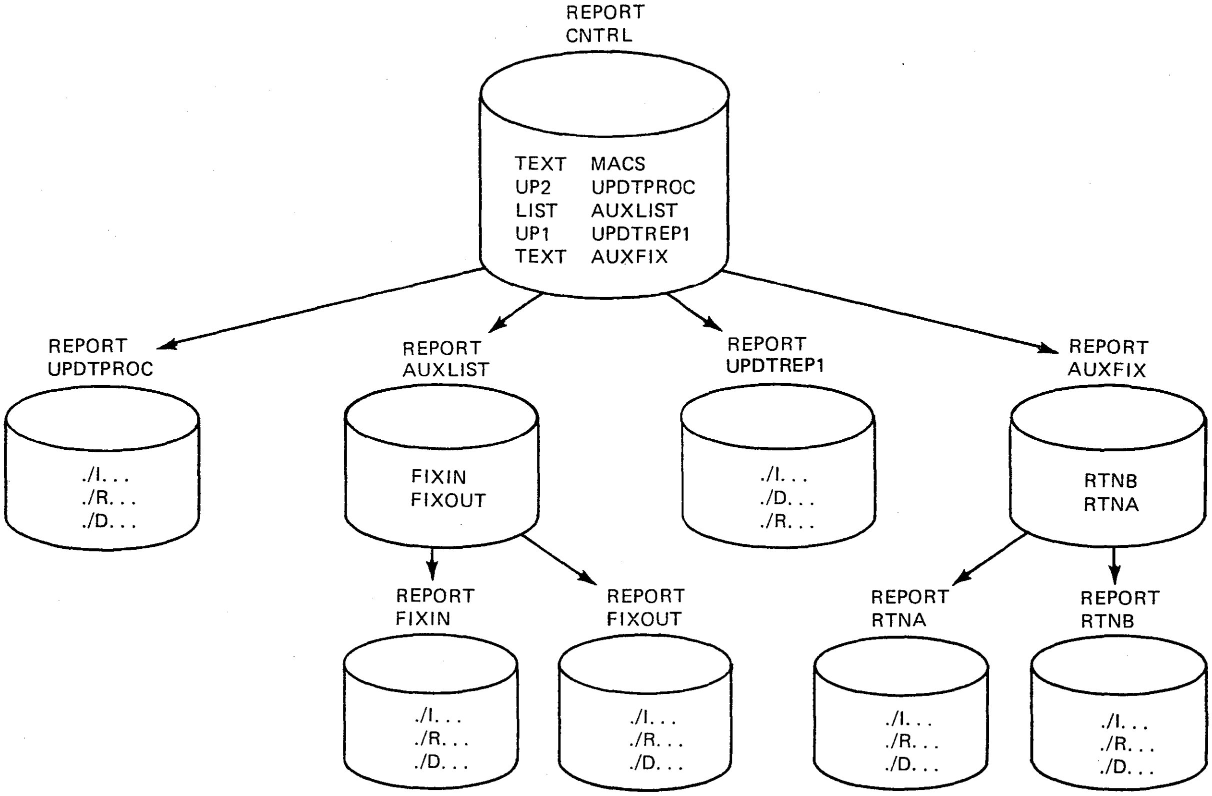
This causes all blanks in the file to be typed as question marks (?) on output.

Appendixes

The following appendixes are provided for your convenience:

- Appendix A: Reserved Filetype Defaults
- Appendix B: VSE/VSAM Functions Not Supported in CMS
- Appendix C: OS/VS Access Method Services and VSAM Functions Not Supported in CMS

Appendix A: Reserved Filetype Defaults



```
update report assemble a (ctl
UPDATING 'REPORT ASSEMBLE A1' WITH 'REPORT RTNA A1'.
UPDATING WITH 'REPORT RTNB A1'.
UPDATING WITH 'REPORT UPDTREP1 A1'.
UPDATING WITH 'REPORT FIXOUT A1'.
UPDATING WITH 'REPORT FIXIN A1'.
UPDATING WITH 'REPORT UPDTPROC A1'.
R;
```

Figure 21. Default EDIT Subcommand Settings for CMS Reserved Filetypes

Appendix B: VSE/VSAM Functions Not Supported in CMS

Refer to the publication Using VSE/VSAM Commands and Macros, SC24-5144, for a description of access method services functions available under DOS/VSE, and, therefore, under CMS. This knowledge of access method services is assumed throughout this publication.

All of VSE/VSAM is supported by CMS, except for the following:

- Non-VSAM data sets with data formats that are not supported by CMS/DOS (for example, BDAM and ISAM files are not supported).
- The SHAREOPTIONS operand has no function in CMS. However, you should specify SHAREOPTIONS 3 in your DEFINE control statement for more efficient operations. When you specify SHAREOPTIONS 3, CMS does not execute the code that attempts to reserve and release system resources.

Appendix C: OS/VS Access Method Services and VSAM Functions Not Supported in CMS

In CMS, an OS user is defined as a user that has not issued the command:

```
SET DOS ON (VSAM)
```

OS users can use all of the access method services functions that are supported by DOS/VSE, with the following exceptions:

- Non-VSAM data sets with data formats that are not supported by CMS/DOS (for example, BDAM and ISAM files are not supported).

- The SHAREOPTIONS operand has no function in CMS. However, you should specify SHAREOPTIONS 3 in your DEFINE control statement for more efficient operation. When you specify SHAREOPTIONS 3, CMS does not execute the code that attempts to reserve and release system resources.

- Do not use the AUTHORIZATION (entrypoint) operand in the DEFINE and ALTER commands unless your own authorization routine exists on the DOS core image library, the private core image library, or in a CMS DOSLIB file. In addition, results are unpredictable if your authorization routine issues an OS SVC instruction.

- The secondary space allocation parameter in the following DEFINE commands is not used by access method services nor DOS/VS VSAM: DEFINE SPACE, DEFINE USERCATALOG, and DEFINE CLUSTER with the UNIQUE parameter. However, you may code this parameter to make your control statement file compatible with an OS/VS VSAM control file.

- The OS access method services GRAPHICS TABLE options and the TEST option of the PARM command are not supported.

- The filename in the FILE (filename) operands is limited to seven characters. If an eighth character is specified, it is ignored.

- The OS access method services CNVTCAT and CHKLIST commands are not supported in DOS/VS access method services. In addition, all OS access method services commands that support the 3850 Mass Storage System are not supported in DOS/VS access method services.

- Figure 22 is a list of OS operands, by control statement, that are not supported by the CMS interface to DOS/VS access method services.

If any of the unsupported operands or commands in Figure 22 are specified, the AMSERV command terminates and displays an appropriate error message.

When you use the PRINT, EXPORT, IMPORT, IMPORTRA, EXPORTRA, and REPRO control statements for sequential access method (SAM) data sets, you must specify the ENVIRONMENT operand with the required DOS options (that is, PRIME DATA DEVICE, BLOCKSIZE, RECORDSIZE, or RECORDFORMAT). You must have previously issued a DLBL for the SAM file.

AMSERV can write SAM data sets only to a CMS disk, but can read them from DOS, OS, or CMS disks.

OS Access Method Services Control Statement	Operands Not Supported in CMS
ALTER	EMPTY/NOEMPTY SCRATCH/NOSCRATCH DESTAGEWAIT/NODESTAGEWAIT STAGE/BIND/CYLINDERFAULT
BLDINDEX	INDATASET OUTDATASET
DEFINE	ALIAS EMPTY/NOEMPTY GENERATIONDATAGROUP PAGESPACE SCRATCH/NOSCRATCH DESTAGEWAIT/NODESTAGEWAIT STAGE/BIND/CYLINDERFAULT TO/FOR/OWNER[1]
DELETE	ALIAS GENERATIONDATAGROUP PAGESPACE SCRATCH/NOSCRATCH
EXPORT	OUTDATASET
IMPORT	INDATASET OUTDATASET IMPORTA
LISTCAT	ALIAS GENERATIONDATAGROUP LEVEL OUTFILE[2] PAGESPACE
PRINT	INDATASET OUTFILE[2]
REPRO	INDATASET OUTDATASET
VERIFY	DATASET

[1] The TO/FOR/OWNER operands are supported for the access method services interface, but are not supported for the DEFINE NONVSAM control statement.

[2] The OUTFILE operand is supported by the access method services interface, but is not supported for the LISTCAT and PRINT control statements.

Figure 22. OS Access Method Services Operands Not Supported in CMS

Index

The entries in this Index are accumulative and reflect the additions of the VM/370 Basic System Extensions Program Product, Program Number 5748-XX8.

IBM System Library Supplement

This Supplement No. SD23-9023-1

Date March 30, 1979

File No. S370-36

For Base Publication GC20-1818-2, *IBM Virtual Machine Facility/370: CMS Command and Macro Reference,* Release 6 PLC 1

Prerequisites None

IBM Virtual Machine Facility/370
Basic System Extensions
Program No. 5748-XX8

This supplement contains replacement pages for VM/370 CMS Command and Macro Reference to support VM/370 Basic System Extensions.

Before inserting any of the attached pages into VM/370 CMS Command and Macro Reference, read carefully the instructions on this cover. They indicate when and how you should insert pages.

Do not insert the attached pages unless you install the program product.

Pages to be Removed	Attached Pages to be Inserted*
Contents vii-x	Contents vii-xi
1-2	1-2.2
7-14	7-14
17-18	17-18
27-32	27-32
41-42	41-42.2
45-56	45-56
63-64	63-64.2
67-68	67-68.2
73-78	73-78
83-84	83-84
87-92	87-92.2
95-100	95-100
105-106	105-106.4
109-116	109-116.2
137-142	137-142
147-152	147-152.2
161-162	161-162
169-170	169-170
173-174	173-174
185-196	185-196.2
199-200	199-200
217-218	217-218
221-224	221-224
227-228	227-228
233-234	233-234
245-246	245-246
249-250	249-250
301-312	301-312
335-336	335-336.4
339-342	339-342
345-348	345-348
Index 349-382	Index 349-382

IBM Corporation, Publications Development, Department D58, Poughkeepsie, New York 12602

Printed in U.S.A.

*If you are inserting pages from different Newsletters/Supplements and _identical_ page numbers are involved, always use the pages with the latest date (shown in the slug at the top of the page). The page with the latest date contains the most complete information.

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

Summary of Amendments

This supplement contains new and updated information in support of VM/370 Basic System Extensions. It contains functions of the initial release and:

- Interactive Help Facility under CMS
- CMS File System Extensions
- CMS/DOS Uplevel to DOS/VSE
- Display Control for the 3270
- Support for the IBM 3289 Model 4 Printer
- Support for the IBM 8809 Tape Unit
- Support for the IBM 3310 and 3370 Direct Access Devices

For a complete list of publications that support VM/370 Basic System Extensions, see _IBM Virtual Machine Facility/370 Basic System Extensions General Information Manual_, GC20-1828.

Note: Please file this cover letter at the back of the base publication to provide a record of changes.