File No. S370-36
Order No. GC20-1804-3

Systems

IBM Virtual Machine Facility/370: Command Language Guide for General Users

I Release 2 PLC 13

This publication provides general users with the basic information they need to use the CP and CMS command languages of VM/370. General users are those users who program, test, and execute applications in a virtual machine.

VM/370 (Virtual Machine Facility/370) is an operating system that manages the resources of a single System/370 computer so that multiple computing systems (virtual machines) appear to exist. VM/370 consists of a Control Program (CP), which manages the real computer, a Conversational Monitor System (CMS), which is a general-purpose conversational time-sharing system that executes in a virtual machine, and a Remote Spooling Communications Subsystem (RSCS), which spools files to and from geographically remote locations.

This publication has two parts and several appendixes. Part 1 is the usage information; it contains the rules for using the command languages and information about virtual disks, CMS files, and virtual machine operation. Part 2 is the reference information; it describes each of the CP and CMS commands for general users. The appendixes include information about the functions of VM/370 commands, problem program debugging, the CMS Batch Facility, CMS macro instructions, and CMS filetype and filemode conventions.

Prerequisite Publications

Introduction to Virtual Storage in System/370, Order No. GR20-4260

IBM Virtual Machine Facility/370: Terminal Users' Guide, Order No. GC20-1810

IBM

Fourth Edition (January 1975)

| This edition, together with Technical Newsletter GN20-2659 dated March
| 31, 1975, corresponds to Release 2 PLC 13 (Program Level Change) of the IBM Virtual Machine Facility/370, and to all subsequent releases unless otherwise indicated in new editions or Technical Newsletters.

Changes are periodically made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 and System/370 Bibliography, Order No. GA22-6822, and its Virtual Storage Supplement, Order No. GC20-0001, for the editions that are applicable and current.

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

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, 24 New England Executive Park, Burlington, Massachusetts 01803. Comments become the property of IBM.

Preface

This publication is an introduction to, and a user's guide and reference manual for, all general users of the VM/370 command languages (CP and CMS).

It contains only those commands available for general users; other commands are listed in the publications IBM Virtual Machine Facility/370: Operator's Guide, Order No. GC20-1806, IBM Virtual Machine Facility/370: System Programmer's Guide, Order No. GC20-1807, and IBM Virtual Machine Facility/370: Remote Spooling Communications Subsystem (RSCS) User's Guide, Order No. GC20-1816.

The publication is organized in two parts: "Part 1: Usage Information" and "Part 2: Reference Information".

Part 1 has six sections:

"Section 1: Introduction and General Concepts" is a summary of general concepts you should be familiar with when you are using VM/370.

"Section 2: The VM/370 CP and CMS Command Languages" describes the VM/370 command environments, the general structure of the command language, and, in general, how to use the features of the VM/370 command languages to help you solve programming problems.

"Section 3: CMS Virtual Disks and How to Use Them" describes the virtual disk system for your virtual machine and how to use it.

"Section 4: The CMS File System" describes the file as it is used in the CMS system. This section gives you information on how to create and name a file, how files are used in the system, and how CMS handles OS data sets and DOS files.

"Section 5: Writing and Executing a Sample Program Using CMS" is an introduction to the VM/370 interactive programming environment. In this section, you are given a program that you can enter at your terminal as you read the book. Together with the information in the four preceding sections, this sample program should provide you with the basic information you need to use your virtual machine and its interactive capabilities.

"Section 6: Virtual Machine Operation" describes the operation of a virtual machine. It includes information about the virtual machine console, attaching devices, handling tape drives, loading operating systems, spooling the console, reading cards, printing, punching, disconnecting the terminal, and alternating execution of operating systems in one virtual machine.

Part 2 is in two sections:

"Section 7: Format and Usage Rules for CMS Commands" and "Section 8: Format and Usage Rules for CP Commands" provide you with all the information you need to use the commands and facilities of VM/370. Also included is a summary of the notational conventions used to describe the commands.

The "Appendixes" include more reference information. They include a list of the functions performed by VM/370 commands, information on how to debug using VM/370, how to use the CMS Batch Facility, a list of macros you can use with the VM/370 assembler, a chart showing the filemodes related to files manipulated by CMS commands, and a list of filetypes reserved by VM/370.

In this publication, the following terminology is used:

- "3330 series" is used to refer to the IBM 3330 Disk Storage Models 1, 2, and 11, and the IBM 3333 Disk Storage and Control Models 1 and 11.
- 2305 refers to the IBM 2305 Fixed Head Storage, Models 1 and 2.
- 3270 refers to the 3277 Display Station.
- Any information pertaining to the IBM 2741 terminal also applies to the IBM 3767 terminal unless otherwise noted.

For a glossary of VM/370 terms see the IBM Virtual Machine Facility/370: Glossary and Master Index, Order No. GC20-1813.

PREREQUISITE PUBLICATIONS

You should be familiar with the information in the publications listed on the front cover before you use this publication.

If the IBM 3767 Communication Terminal is a virtual console, the publication IBM 3767 Operator's Guide, Order No. GA18-2000, is also a prerequisite.

COREQUISITE PUBLICATIONS

The information in the following publications describes certain VM/370 facilities in greater detail and with more examples than can be contained in this publication:

IBM Virtual Machine Facility/370:

EDIT Guide, Order No. GC20-1805

EXEC User's Guide, Order No. GC20-1812

System Programmer's Guide, Order No. GC20-1807

System Messages, Order No. GC20-1808

Glossary and Master Index, Order No. GC20-1813

Remote Spooling Communications Subsystem (RSCS) User's Guide, Order No. GC20-1816

OS/VS-VM/370 Assembler Programmer's Guide, Order No. GC33-4021

References in text to titles of prerequisite and corequisite VM/370 publications will be given in abbreviated form.

Summary of Amendments
for GC20-1804-3
VM/370 Release 2 PLC 13

VM/370 MEASUREMENT FACILITY

New: Program Feature

A new class G command, INDICATE, allows you to display, at your terminal, certain system load conditions and your virtual machine's use of system resources. The following changes to this manual reflect this support:

- "Section 8: Format and Usage Rules for CP Commands" is updated to include the new INDICATE command.
- "Figure 34. Commands Accepted from Each User Class" is updated.
- "Figure 35. CP Command Summary" is updated.
- "Figure 44. Commands for System and Hardware Analysis" is updated.

VM/VS HANDSHAKING FEATURE

New: Programming Feature

The VM/VS Handshaking feature is a communication path between VM/370 and OS/VS1 that makes each system control program aware of certain capabilities and requirements of the other. The following changes to this manual reflect this support:

- A new operand, PAGEX, is added to the CP SET command.
- The response for the CP QUERY SET command line is updated.
- "Figure 34. Commands Accepted from EAch User Class" is updated.
- "Figure 41 Commands for Virtual Machine Control is updated.

IBM 3270 REMOTE SUPPORT

New: Program Feature

VM/370 now supports the IBM 3270 Information Display System as a remote virtual machine console attached via nonswitched point-to-point lines to a 2701 Data Adapter Unit, 2703 Transmission Control Unit, or 3704/3705. Communications Controller in emulation mode. The remote 3270 user also has the capability of copying an entire screen display on a 3284, 3286, or 3288 Printer at the remote location.

The following changes to this manual reflect this new support:

- "Section 7. Format and Usage Rules for CMS Commands" is updated to reflect additional operands and subcommands of the EDIT command.
- "Section 8. Format and Usage Rules for CP Commands" is updated to include the COPY function of the SET command.
- "Figure 34. Commands Accepted from Each User Class", "Figure 36. Commands to Control a Terminal Session", and "Figure 42. Commands to Control VM/370" are updated to reflect additional operands to the NETWORK, QUERY, and SET commands.

NEW OPERANDS FOR SET COMMAND

New: Program Feature

Two new operands to the SET command described in "Section 8: Format and Usage Rules for CP Commands" allow the virtual machine user to enable and disable the ECMODE and/or ISAM options, dynamically.

NEW OPERAND FOR SLEEP COMMAND

New: Program feature

A delay time interval may now be specified as an operand on the SLEEP command line. The use of this optional operand causes the virtual machine to automatically awaken at the end of the specified time interval. The new command format is described in "Section 8: Format and Usage Rules for CP Commands".

MISCELLANEOUS

Changed: Documentation Only

This Technical Newsletter contains other minor technical and typographic changes.

NEW DEVICE SUPPORT

New: Program Feature

The IBM 3340 Direct Access Storage Facility is now supported by VM/370. This support includes Rotational Position Sensing (RPS) and the 3348 Data Module, Models 35, 70 and 70F (supported as a Model 70). The following changes to this manual reflect this support:

- The control statements and messages of the DASD Dump Restore program are updated.
- T3340 is a new operand for the CP DEFINE command.
- The device type field in the response for the CP QUERY VIRTUAL DASD command line is updated.

VM/370 supports the IBM 3767 Communication Terminal (at 300 bps) operating as an IBM 2741 Communication Terminal only. No coding changes were necessary; the following changes to this manual reflect this support:

- The list of prerequisite publications in the Preface is updated.
- "The Terminal: Your Virtual Console" discussion in "Section 5: Writing and Executing a Program Using VM/370" is updated.
- The discussions of the BLIP and AUTOREAD functions of the CMS SET command are updated.
- The discussion of the APL operand of the CP TERMINAL command is updated.

READING DOS FILES FROM DOS DISKS

New: Programming Feature

Using OS macros, CMS can now read DOS files that reside on DOS disks. No DOS macros are simulated. The following changes to this manual reflect this support:

- "Section 4: The CMS File System" is updated.
- The CMS ACCESS, FILEDEF, LISTDS, MOVEFILE, and STATE command descriptions in "Section 7: Format and Usage Rules for CMS Commands" are updated.
- "Figure 37. Commands to Develop Programs and Process Data" is updated.

CMS ZAP SERVICE PROGRAM ENHANCED

New: Program Feature

The CMS ZAP service program is enhanced so that it can now modify or dump TXTLIB and MODULE files in addition to LOADLIB files. "Figure 13. CMS Command Summary" is updated to reflect this support. This service program is intended for use by system support personnel only.

NEW SUBSYSTEM SUPPORTED AS VM/370 COMPONENT

New: Component

The Remote Spooling Communications Subsystem (RSCS) is now supported as a VM/370 component. RSCS transmits spool files across a teleprocessing network to and from remote stations. Remote stations supported include HASP- and ASP-type batch processors and work stations. The new component is a control program designed to run in a virtual machine dedicated to remote spooling. The following changes reflect this support:

- The Preface is updated.
- "Section 1: Introduction and General Concepts" is updated.
- "Figure 34. Commands Accepted from Each Privilege Class" is updated.
- "Figure 35. CP Command Summary" is updated.
- The CP CHANGE, DEFINE, DETACH, ORDER, SET, SPOOL, and TRANSFER command descriptions are updated.

- A new CP command, TAG, is described.
- "Figure 36. Commands to Control a Terminal Session" is updated.
- "Figure 43. Commands for Spooling Control" is updated.
- "Appenidx C: Using the CMS Batch Facility" is updated.

ENHANCEMENT FOR THE VIRTUAL MACHINE

New: Program feature

Programs using block multiplexer channel operations (for example, DOS/VS, VS1, and VS2) can now be executed in block multiplexer mode on the virtual machine. This virtual machine option may increase the throughput of the virtual machine's operating system because it allows the overlap of SIOs to virtual devices without encountering the channel busy condition. The following changes to this publication reflect this support:

- "Figure 34. Commands Accepted from Each Privilege Class" is updated.
- The CP DEFINE and QUERY commands have a new operand, CHANNELS.

MISCELLANEOUS

Changed: Documentation Only

Numerous minor editorial changes have been made; these are too numerous to mention. However, a list of the major editorial changes and all miscellaneous technical changes follows.

"Part 1: Usage Information" has many editorial changes. In addition, several sections about the operation of virtual machines and CMS, which were in the VM/370: Planning and System Generation Guide, are now included in Part 1:

- Additional information about the terminal mode of operation, attention interrupts, and the CMS command search order is included in "Section 2: The VM/370 CP and CMS Command Languages."
- Additional information about accessing CMS disks is added to "Section 3: CMS Virtual Disks and How to Use Them."
- Additional information about compiling programs under CMS and tape handling in CMS is included in "Section 4: The CMS File System."
- Additional information about entering the continuation character in column 72 is included in "Section 5: Writing and Executing a Sample Program Using CMS."
- A new section, "Section 6: Virtual Machine Operation" is added to Part 1.

"Part 2: Reference Information" has the following changes:

- The "Notational Conventions" section is expanded.
- "Figure 13. CMS Command Summary" contains editorial changes.
- The following CMS command descriptions are clarified:

  -ACCESS
  -ASSEMBLE
  -COMPARE
  -COPYFILE
  -DDR control statements
  -EDIT
  -ERASE
  -EXEC
  -FILEDEF
  -FORMAT
  -GENDIRT
  -GENMOD
  -GLOBAL
  -INCLUDE (Two new options, DUP and NODUP, are described)
  -LISTFILE
  -LOAD (Two new options, DUP and NODUP, are described)
  -PRINT
  -PUNCH
  -QUERY
  -READCARD
  -RELEASE
  -RENAME
  -RUN
  -SET
  -START
  -STATE
  -SVCTRACE
  -SYNONYM
  -TAPE
  -TAPPDS
  -TXTLIB
  -TYPE
  -UPDATE

- "Figure 17. Summary of EDIT Subcommands and Macros" contains editorial changes.
- Two previously undocumented immediate commands RO and SO are now included. SO temporarily suspends the recording of trace information and RO restarts tracing that was suspended via the SO command.

- "Figure 33. CP Privilege Class Descriptions" is updated.
- "Figure 34. Commands Accepted from Each Privilege Class" is reformatted.
- "Figure 35. CP Command Summary" is updated.
- The following CP command descriptions are clarified:

  -#CP
  -CLOSE
  -CP
  -DEFINE
  -DISCONN
  -DISPLAY
  -DUMP
  -EXTERNAL
  -IPL
  -LINK
  -LOGOFF
  -LOGON
  -MESSAGE
  -QUERY
  -SET
  -TERMINAL
  -TRACE
  -TRANSFER

- The RDTAPE and PRINTL macro descriptions in "Appendix D: CMS Macro Instructions" are clarified.
- "Figure 48. Reserved Filetypes" is updated.

CMS EDITOR LINE RENUMBERING FEATURE

New: Program Feature

A new CMS Editor subcommand recomputes the line numbers of VSBASIC and FREEFORT source files. "Figure 13. Summary of EDIT Subcommands" and the EDIT command error messages in "Section 6: Format and Usage Rules for CMS Commands" were changed to support this feature.

IBM 3704/3705 COMMUNICATIONS CONTROLLERS NETWORK CONTROL PROGRAM (NCP) AND PARTITIONED EMULATION PROGRAM (PEP) SUPPORT

New: Program Feature

VM/370 now support all three of the 3704/3705 control programs:

- Emulation Program (EP)
- Network Control Program (NCP)
- Partitioned Emulation Program (PEP)

The following changes reflect this support

- The Preface is updated to remove the statement limiting support of the 3704/3705.
- "Figure 9. CMS Command Summary" in "Section 6: Format and Usage Rules for CMS Commands" is updated.
- "Figure 31. CP Command Summary" in "Section 7: Format and Usage Rules for CP Commands" is updated.
- The DIAL, DISCONN, LOGON and LOGOFF descriptions in "Section 7: Format and Usage Rules for CP Commands" are updated.
- "Figure 38. Commands to Control VM/370" in "Appendix A: Functions of VM/370 Commands" is updated.
- "Figure 43. Disk Determination" in "Appendix E: Disk Determination (Filemode Management)" is updated.
- "Figure 44. Reserved Filetypes" in "Appendix F: Reserved Filetype Descriptions" is updated.

MISCELLANEOUS

Changed: Documentation Only

- The running foot for section 7 is corrected on the TNL pages to read "Section 7: Format and Usage Rules for CP Commands"
- The Index is corrected.

CHANGES TO GENERAL INFORMATION IN PART 1

New: Documentation Change

Part 1 of the publication has been changed to include information on general concepts for VM/370 users, on the CP and CMS command languages, on the CMS disk system, and on the CMS file system. Also, there is an example program for use as an introduction to the interactive processing capabilities of VM/370.

GENERAL CHANGES TO PUBLICATION

Changed: Documentation expanded

Editorial changes have been made throughout the publication to clarify usage.

New: Documentation Change

In Section 6, "Format and Usage Rules for CMS Commands", the error messages and return codes have been included in the command descriptions.

CHANGES TO CMS COMMANDS

Changed: Documentation Expanded

SYNONYM COMMAND

Updated to charify use of the command.

Changed: Program Feature

ACCESS COMMAND

Updated to clarify its use with OS Read/Only disk.

DDR COMMAND

Updated to support 9-track 6250 bpi tapes and 3330 Model 11 disks.

EDIT COMMAND

Updated to support new subcommands for graphic display devices.

FILEDEF COMMAND

Updated to reflect new support for dynamic entry of a data set name and for new options MEMBER and CONCAT. Also, 6250 bpi tapes are now supported by FILEDEF.

GLOBAL COMMAND

Updated to support use of the new FILEDEF CONCAT option.

MOVEFILE COMMAND

Updated to include support for accessing OS data sets.

QUERY COMMAND

Updated to include support for querying the status of OS disks accessed by CMS.

SET COMMAND

Updated to include support for setting conditions for use with a display terminal.

STATE COMMAND

Updated to include support for verifying the existence of an OS disk accessed by CMS.

TAPE COMMAND

Updated to include support for 6250 bpi tapes.

UPDATE COMMAND

Updated to allow flexibility in format specification of auxiliary files.

New: Program Feature

ASSEMBLE COMMAND

Updated to reflect the new options for the assembler and to clarify use of the command.

LISTDS COMMAND

Added to the system: Lists information about data sets on OS disks accessed by CMS.

CHANGES TO CP COMMANDS

Changed: Documentation Expanded

* (asterisk) COMMAND

Updated to clarify its use.

ADSTOP COMMAND

Updated to clarify use of the VM/370 virtual machine assist feature with ADSTOP.

CHANGE COMMAND

Updated to clarify use.

CLOSE COMMAND

Updated to clarify use.

DISCONN COMMAND

Updated to clarify use.

IPL COMMAND

Updated to clarify use.

LOGOFF COMMAND

Updated to clarify use.

LOGON COMMAND

Updated to clarify use.

REWIND COMMAND

Updated to clarify use.

SET COMMAND

Updated to clarify use.

SLEEP COMMAND

Updated to clarify use.

TRACE COMMAND

Updated to clarify use.

Changed: Program Feature

DEFINE COMMAND

Updated to include support for the GRAF operand. GRAF allows you to define a temporary graphic device for your virtual machine.

DISPLAY COMMAND

Updated to allow you to specify a range of registers or storage.

DUMP COMMAND

Updated to allow you to specify a range of registers or storage.

MESSAGE COMMAND

Name of the MSG command has been changed to MESSAGE.

QUERY COMMAND

Updated to support GRAF, and PFnn operands. These new operands allow you to query the status of console spool files, display terminals, and program function keys, respectively.

SET COMMAND

Updated to include support for the ASSIST and PFnn options. ASSIST invokes the virtual machine assist feature; PFnn allows you to define functions associated with program function keys on a graphic display terminal.

SPOOL COMMAND

Updated to support the * (asterisk) character option. * specifies a universal spooling class which allows your virtual reader to read all classes of spool files.

Also, the CLOSE, FOR, and PURGE options are supported.

TERMINAL COMMAND

Updated to clarify use. Also, updated to reflect support for the MODE operand, which allows you to set the mode of your terminal to either CP mode (Control Program) or VM (virtual machine).

TRANSFER COMMAND

Updated to support the FROM userid operand, which allows you to reclaim a file that you spooled to another user.

New: Program Feature

#CP COMMAND

Added to allow you to enter CP commands from a virtual machine environment without issuing an interrupt.

ATTN COMMAND

Added to allow an attention interrupt to be entered at the keyboard.

CP COMMAND

Added to allow entry of the characters "CP" while processing in the CP command environment.

REQUEST COMMAND

Added to allow you to enter an attention interrupt at the keyboard of you virtual console.

Contents

Figures

Part 1: Usage Information

This part of the publication contains rules and information to help you use all of the facilities of VM/370. It contains six sections.

"Section 1: Introduction and General Concepts" is a summary of general concepts you should be familiar with when you are using VM/370.

"Section 2: The VM/370 CP and CMS Command Languages" describes the VM/370 command environments, the general structure of the command language, and how you can use the features of the VM/370 command languages to aid you in solving programming problems.

"Section 3: CMS Virtual Disks and How to Use Them" describes the virtual disk system for your virtual machine and how to use it.

"Section 4: The CMS File System" describes the file as it is used in the CMS system. This section gives you information on how to create and name a file, how these files are used in the system, and how CMS handles OS data sets and DOS files.

"Section 5: Writing and Executing a Sample Program Using CMS" is an introduction to the VM/370 interactive programming environment. In this introductory section, you are given a program that you can enter at your terminal as you read the book. Together with the information in the four preceding sections, this sample program should provide you with the basic information you need to use your virtual machine and its interactive capabilities.

"Section 6: Virtual Machine Operation" contains additional information about operating virtual machines. This information generally applies to non-CMS virtual machines and includes an example of alternately executing CMS and another operating system in the same virtual machine.

This section is a summary of general concepts that are useful to you when you are using VM/370. It is a prologue to the following sections, which describe how to use virtual disks, and how to use the CMS file system.

Virtual Machine Facility/370 (VM/370) is a system control program 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, the CP, CMS, and RSCS Command Languages. Using your terminal (as a virtual console) and the command languages, you have many of the capabilities of the operator of a real machine.

VM/370 Control Program (CP), the Conversational Monitor System (CMS), and the Remote Spooling Communications Subsystem (RSCS)

VM/370 has three components: the Control Program (CP), the Conversational Monitor System (CMS), and the Remote Spooling Communications Subsystem (RSCS). CP controls the resources of the real machine; that is, the physical machine in your computer room. CMS is the conversational operating system designed specifically to run under CP. RSCS is a subsystem designed to supervise transmission of files across a teleprocessing network controlled by CP. This section describes CP and CMS; for information about RSCS, see the VM/370: Remote Spooling Communications Subsystem (RSCS) User's Guide.

THE VIRTUAL MACHINE

The virtual machine is a software counterpart of a real machine. Its control program can be any of those supported by VM/370. These are listed in the VM/370: Introduction.

The virtual machine components are the virtual system console (your terminal), a virtual CPU, virtual storage, and virtual channels and input/output (I/O) devices.

Your virtual machine is a logical extension of the real computer. The virtual machines are defined in the VM/370 directory, which is a list of all the virtual machines logically extended from the real machine. Each entry in the VM/370 directory lists the configuration of a virtual machine.

Your Virtual Disks

A virtual disk is a logical subdivision of a real disk. For practical purposes, it is the same as a real disk on a real machine. Virtual disks

have their own device addresses and cylinder numbers. Each CMS virtual disk has a Master File Directory that lists each of the files contained on it.

The size of your virtual disks is defined in the VM/370 directory entry for your virtual machine. The virtual disk can be equal to or less than the size of the real disk. For CMS, the maximum size of a virtual disk is less than the real disk and varies with the device type. If you need more space, your system programmer can give it to you by redefining the number of cylinders, by defining more disks in your VM/370 directory entry; or you can assign temporary disk (T-disk) space for your virtual machine. With CMS, you can have up to ten virtual disks logically attached to your virtual machine. These ten disks are called the A, B, C, D, E, F, G, S, Y, and Z disks and are described in more detail in Section 3.

VM/370 Files

The file is the essential data unit in the VM/370 system. Programs, program input, procedures, and any other type of data unit you want to use on the system, are files in the VM/370 system.

In CMS, a file is a logically related group of records that you create. Using the EDIT command, you can create and modify a file yourself in the CMS command environment; using FILEDEF, you can define OS data sets and DOS files as files for use under CMS.

Spooling in VM/370

On a real machine, spooling is, in general, the process of transmitting a unit record file from the CPU to a peripheral device for processing or from a device to the CPU for processing. When the file is sent to the peripheral device, the CPU is free for more work.

The concept of spooling also applies to virtual unit record I/O devices. You process virtual unit record files in your virtual machine and, while they are processing, transmit them to your virtual printer or to your virtual punch. Your virtual machine is then available for more work. This collection of data directed to your virtual printer or punch is called a spool file. The output spool file can be transmitted to its counterpart on the real machine, to your own reader, or to the reader of another virtual machine. Thus, the spool file is waiting to be processed by a real device or another virtual machine.

VM/370 expands the concept of spooling to include the creation of a spooled console log which is a record of all of your virtual console input/output, including CP commands and responses. This console spool file can be sent to the real printer when you want a copy of it.

The Remote Spooling Communications Subsystem (RSCS) lets you transmit files across a teleprocessing network to geographically remote locations via the CP TAG and SPOOL commands. For details on how the network is structured and operated, see the VM/370: Remote Spooling Communications Subsystem (RSCS) User's Guide.

THE CP AND CMS COMMAND LANGUAGES

You control your virtual machine using the CP and CMS command languages. Any virtual machine can use the CP command language; you can use the CMS command language only if your virtual machine operating system is CMS.

The CP Command Language

The CP command language gives you control of the devices attached to your virtual machine. Many of the functions of a real computer console are simulated via the CP command language. Also, there are commands that provide spooling and disk-sharing capabilities.

CP commands fall into one or more privilege classes; this publication provides information on only those commands that can be used by the general user (class G) or any user (class any). General users are those users who program, test, and execute applications on a virtual machine. The CP command privilege classes available to your virtual machine are defined in your VM/370 directory entry.

Using CP commands, you can send messages to the VM/370 system operator and other users, modify the configuration of your virtual machine, and use the virtual machine input/output devices. CP commands are available to all virtual machines under VM/370. You can invoke these commands when you are in the virtual machine environment using CMS (or some other operating system) in your virtual machine.

The CMS Command Language

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

Many language processors can be executed under CMS: the assembler, VS BASIC, OS FORTRAN IV, OS COBOL, and OS PL/I Optimizing and Checkout Compilers. You can find a complete list of language processors that can be executed under CMS in the VM/370: Introduction. CMS executes the assembler and the compilers when you invoke them with CMS commands. The ASSEMBLE command is described in this manual; the supported compiler commands are described in the appropriate Program Product documentation.

The CMS EDIT command allows you to create and modify files; the EXEC command helps you to create a procedure consisting of CP and CMS commands, which has the conditional execution capability of a macro language; the DEBUG command gives you several program debugging subcommands.

Other 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.

Since you can invoke CP commands from within the CMS virtual machine environment, the CP and CMS command languages are, for practical purposes, a single, integrated command language for CMS users.

This section describes the VM/370 command environments, the general structure of the CP and CMS command languages, the user-defined commands, command abbreviations, truncations, synonyms, and the command search order.

The VM/370 system is comprised of the Control Program (CP) and the Conversational Monitor System (CMS). CP controls the real machine. You can use CP commands to manipulate your virtual machine. CMS is an interactive system designed specifically for use with CP for problem solving and program development. CMS gives you a file system and the commands required to manipulate files, and the virtual disks on which they reside.

There are two types of VM/370 commands: system commands and user-defined commands. The system commands are those defined by the CP and CMS command languages. User-defined CMS commands are those you create yourself using the EXEC command or the LOAD and GENMOD commands. User-defined CP commands are also allowed; your installation system programmer must create them. The procedure is described in the VM/370: System Programmer's Guide.

Depending on the way you are using the system, you may want to be in a specific command environment. A command environment is the environment of your virtual machine as defined by the command language subset you are using. If you are using the CP command language, you are in the CP command environment; if you are using CMS or another operating system, you are in the virtual machine command environment.

Many commands can also have abbreviations and minimum truncations, as defined by the system, and can have synonyms, which you define.

Since there are many types of commands, VM/370 provides a set of conventions for searching through the system for a given command. For example, you may have defined your own command with the same name as a CMS system command, therefore, you must know which command actually executes. The rules governing the search for the correct command are contained in this section.

VM/370 Command Environments

There are two basic command environments: the CP command environment and the virtual machine command environment.

The CP command environment is reached when you log on to VM/370 and issue CP commands. Depending on the task you want to perform on your virtual machine, you may want to switch in and out of this environment. The means for entering and exiting from this environment are described in "Interrupting the Execution of a Command" in this section.

You can find out which command environment you are in by entering a null line (that is, pressing the Enter key, or equivalent, with no data). VM/370 responds to a null line by displaying the current command environment.

The virtual machine command environment is the environment you reach when you load your virtual machine with the CMS operating system or another operating system.

When you load CMS in your virtual machine, you can use both the CP and CMS command languages. The CMS command environment has three subcommand environments that are entered by means of the EDIT, EXEC, and DEBUG commands.

The EDIT command places your virtual machine in the EDIT subcommand environment, where 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, the EDIT mode or the INPUT mode. EDIT mode lets you modify a file; INPUT mode lets you create or add to a file.

The EXEC subcommand environment is entered via the EXEC command. In this environment, you execute procedures that contain combinations of CP and CMS commands to perform the functions you specify. These procedures are usually created in the EDIT environment.

The DEBUG command places your virtual machine in the DEBUG subcommand environment, in which you can issue commands to display registers and storage, specify breakpoints (address instruction stops), display the contents of control words, and so on.

How to Enter CP and CMS Commands

A VM/370 command consists of a command name, usually followed by one or more positional operands and, in some cases, by an option list. The general form for the command line is:

```
| command name | [operand...] [(option...[)]]
```

You must use one or more blanks to separate each entry in the command line unless otherwise indicated.

THE COMMAND NAME

The command name is an alphameric symbol of not more than eight characters. In general, the names are verbs which describe the function you want the system to perform. For example, you may want to find out whether a certain user is logged on the VM/370 system. In this case, you would use the CP QUERY command.

THE COMMAND OPERANDS

The command operands are keywords and positional operands of no more than eight alphameric characters each. The operands specify the information on which the system operates when it performs the command function. For the QUERY command, for example, you could use the USER or userid operand to find out whether the user is on the system.

Some commands require no operands; others require several. You can find each class G, class Any, and CMS command with all of its operand requirements in Part 2 of this publication.

You must write the operands in the order in which they appear in the command formats in Part 2, unless otherwise specified. When you are using CMS, blanks should be used to separate the last operand and the option list.

THE COMMAND OPTIONS

The command options are keywords used to control the execution of the command. The command formats in Part 2 show all the options for each command.

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

If conflicting or duplicate options are entered, the last entered is the option in effect for the command. Exceptions to this rule are the CMS COPYFILE, FILEDEF, ERASE, and FORMAT commands. See the individual descriptions of these commands in Part 2 for more information.

COMMENTS IN THE CP AND CMS COMMAND LANGUAGES

You can write comments with CP commands of the following types:

- Commands with no operands
- Commands with a fixed number of operands
- Commands with a single optional operand

You cannot write comments with commands that have a variable number of operands or with commands that have more than one optional operand.

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 enter comments on your console by using the CP * command.

CHARACTER SET USAGE

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

Character Set	Names	Symbols
Separator	Blank	
National	Dollar Sign	$
	Pound Sign	#
	At Sign	@
Alphabetic	Upper Case	A - Z
	Lower Case	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

How to Write User-Defined Commands

You can update existing commands and create your own commands using the CMS EXEC command and the LOAD and GENMOD commands.

EXEC COMMANDS

The EXEC command allows you to create an EXEC procedure that has the effect of a command and which can be invoked by its filename, just as a command is invoked. EXEC procedures can contain CP, CMS, and user-defined commands and provide a convenient way of generating a predefined sequence of commands. You are implicitly specifying the EXEC command when you issue the filename of the EXEC procedure.

There is one case when you must specify the EXEC command and its associated filename explicitly: when an EXEC is nested within another EXEC, the nested EXEC must be specified explicitly.

Certain EXEC control statements let you conditionally execute statements within the procedure. You conditionally execute statements by means of the &IF, &LOOP, and &GOTO statements, similar to the conditional and looping facilities found in high level languages.

You can find all the information you need to write EXEC commands in the VM/370: EXEC User's Guide.

A special type of EXEC procedure called a PROFILE EXEC can be used to set up a predetermined operating environment within CMS each time you use your virtual machine. If you have a PROFILE EXEC it is executed before the first command after you IPL CMS. It defines the conditions you want while you are programming under CMS. For example, if you are an assembler language programmer, you need the CMS, TEXT, and OS macro libraries accessed while you are processing. In your PROFILE EXEC, then, you could issue a GLOBAL command to access these libraries automatically whenever you IPL the CMS operating system.

COMMANDS CREATED BY LOAD AND GENMOD

You can use the LOAD and GENMOD commands to create program modules. A module is a relocatable file whose external references have been resolved; that is, a module is a file that you can execute by invoking its filename. The module filename is, effectively, a CMS command you define for your own use.

A source program must be assembled or compiled to produce an object file that has a filetype of TEXT. The LOAD command loads the object file into virtual storage in your virtual machine. When LOAD executes, all external references for the file are resolved. You then use GENMOD to create an absolute core-image of the file or files loaded in storage. Its filetype is always MODULE. This core-image copy may then be used as a CMS command by invoking its filename.

If you are creating a module containing more than one TEXT file, you include the additional TEXT files with the INCLUDE command, then issue the GENMOD.

Truncating and Abbreviating Commands

To make the entering of commands on the keyboard more convenient, CP and CMS allow many commands (as well as many operands and options) to be used in a shortened form. The shortened form can be either an abbreviation or a truncation to its minimum form. Minimum truncations are shown within the format box and abbreviations are shown with the description that follows the format box for all the command formats in Part 2.

In those formats, the minimum truncation is shown in uppercase letters in the command format box; the optional letters are shown in lowercase. This rule holds for options and operands as well. A good example of command truncation is the EDIT command. The format shows the command name as "Edit". This means you can enter the EDIT command in any of the following forms:

```
EDIT
EDI
ED
E
```

E is the minimum truncation, but CMS accepts all of the above forms for the command.

Abbreviations are shorter forms of commands and operands. Abbreviations for commands are shown below the full name of the command in the format box. Abbreviations for operands and options are shown in the description of the individual operands and options that follow the format box. For example, the operand READER has both a minimum truncation and an abbreviation. In the format box, it is shown as:

```
Reader
```

indicating the minimum truncation is R. In the discussion of the reader operand that follows, it is shown as:

```
READER
RDR
```

indicating that the abbreviation is RDR.

Synonyms for CMS Commands

Using the EDIT command, you can create a table of synonyms for both CMS commands and commands you define yourself. You create the file using the EDIT command in the form

```
EDIT filename SYNONYM
```

(the filetype must be SYNONYM). You then enter the records that will be in the table, in the form

```
commandname synonymname count
```

where "commandname" is the name of the command for which you are creating a synonym, "synonymname" is the name of the synonym for the command name, and "count" is the number of characters you want to use as a minimum truncation value for the synonym.

In order to use the synonym table once you have created it and filed it in your system, you must invoke the SYNONYM command at the beginning of every terminal session in which you want to use the table.

You can find a more detailed explanation of how to use the SYNONYM command in Section 7.

CMS Command Search Order

You can create a user-defined command (that is, a module or EXEC procedure) which executes in your virtual machine and resides on disk. To execute such a command or EXEC procedure, you only have to enter the filename from the terminal. However, be aware of the CMS search order for terminal input. Once a match is found, the search stops. The search order is:

1. EXEC file on any currently accessed disk. (The CMS RUN command is an EXEC file.)
2. Valid abbreviation or truncation for an EXEC file on any currently accessed disk.
3. Nucleus resident or transient area command. (The nucleus resident CMS commands are: CP, DEBUG, ERASE, GENMOD, INCLUDE, LOAD, LOADMOD, START, and STATE. The transient commands are: ACCESS, COMPARE, DISK, FILEDEF, GENDIRT, GLOBAL, LISTFILE, MODMAP, PRINT, PUNCH, QUERY, READCARD, RELEASE, RENAME, SET, SVCTRACE, SYNONYM, TAPE, and TYPE.)
4. Command module on any currently accessed disk. (All the remaining CMS commands are disk resident and execute in the user area.)
5. Valid abbreviation or truncation for nucleus-resident or transient area command module.
6. Valid abbreviation or truncation for disk resident command.

For example, if you create an EXEC file with the same name as a disk resident command, the CMS search always finds the EXEC file first. Thus, the disk resident command is never executed.

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

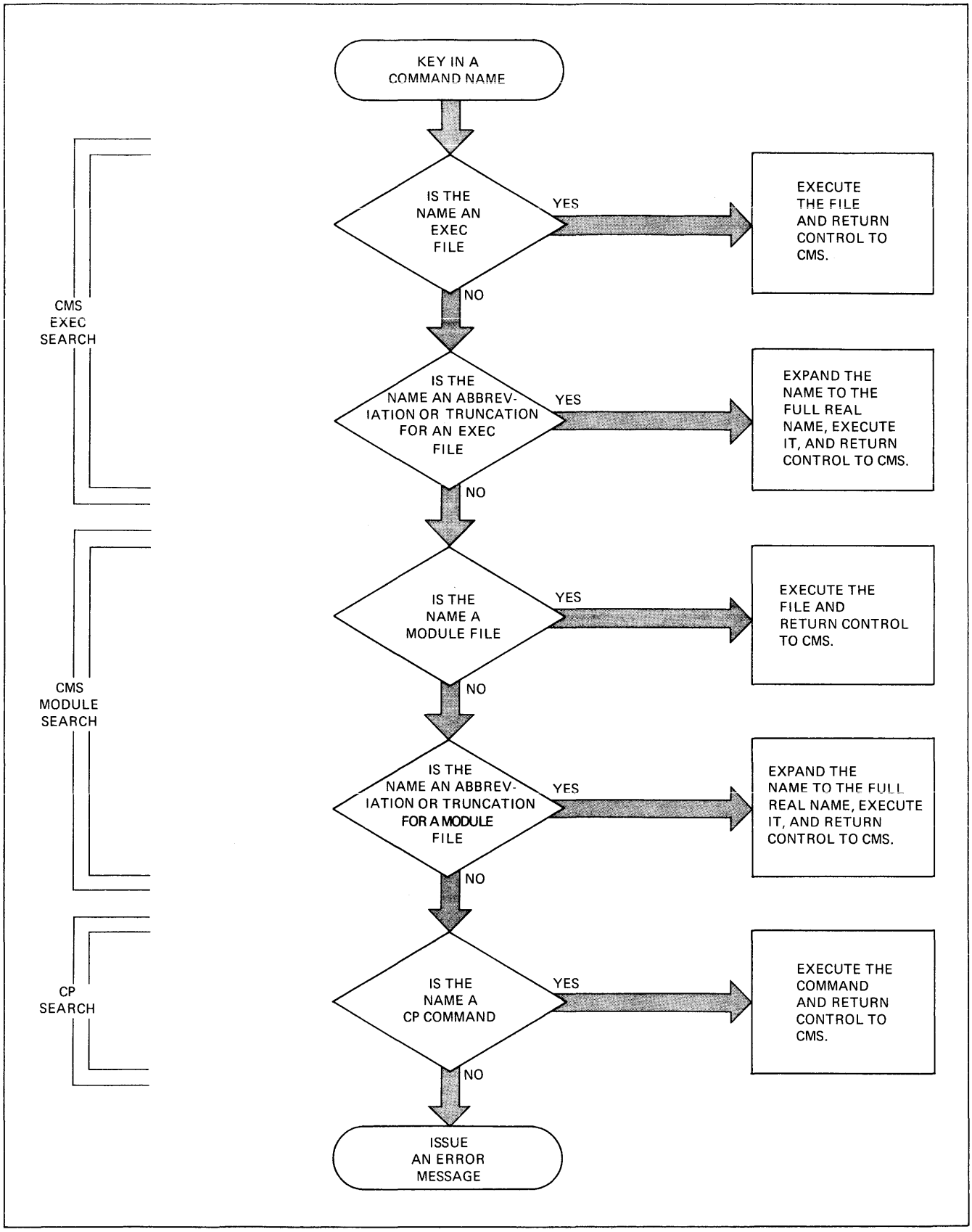


Figure 2. How CMS Searches for the Command to Execute

INTERRUPTING THE EXECUTION OF A COMMAND

You can suspend the execution of a command in your virtual machine using the Attention key (or its equivalent) on your terminal keyboard. Functionally, the Attention key requests the Control Program to interrupt the command currently executing, thus allowing you to key in new input. The interrupt takes effect immediately in most cases, but some commands complete execution before the interrupt takes effect.

The name and physical location of the Attention key varies with different styles of terminals supported by VM/370. See the VM/370: Terminal User's Guide for details on various styles of terminals.

Using the Attention Key

Using the Attention key, you can switch command environments (that is, switch from the CP to the VM command environment or vice versa), at the same time you request an interrupt. What happens when you press the Attention key depends on two factors: the command environment of your virtual machine and the mode setting of your virtual console.

Your virtual machine can be in one of two command environments: the CP command environment (CP) or the Virtual Machine command environment (VM). These environments are defined by usage. When you IPL an operating system for your virtual machine, your virtual machine is in the VM environment, otherwise, it is in the CP environment.

You set the mode of your virtual console using the CP TERMINAL command. The MODE operand of this command allows you to set your virtual console in either the CP or VM mode. These modes correspond, in general, to the preceding command environments description. Each of the terminals that can be used with the VM/370 system has a key that is the equivalent of the Attention key on the 2741 (with which you signal an attention interrupt). Unless otherwise noted, where the term "Attention key" is used in this publication, the phrase "(or equivalent)" is implied. The equivalent key on the 1050 terminal is the RESET LINE key; on the 3270 terminal, the Enter key.

On a 3270 terminal you can signal an attention interrupt to the virtual machine with the ATTN or REQUEST commands. Issuing the ATTN or REQUEST commands on a 3270 is equivalent to pressing the Attention key once on a 2741. On a 3270 terminal, pressing the Enter key is equivalent to pressing the Attention key twice quickly on a 2741.

For a 3215, you press the Attention key once to signal an attention interrupt to the virtual machine, then enter the #CP command to get to CP mode.

How CP interprets attention interrupts issued by the virtual machine users depends on whether the terminal mode is set to CP or VM.

The default mode setting for the primary system operator is:

```
TERMINAL MODE CP
```

If your default terminal mode is CP, or if you issued the above command, pressing the Attention key one or more times forces your virtual machine into CP mode.

For all other users, the default mode setting is:

```
TERMINAL MODE VM
```

If the terminal mode is VM, pressing the Attention key once passes an interrupt pending condition to the virtual machine operating system. Pressing the Attention key twice quickly (or pressing the 3270's Enter key) places your virtual machine in CP mode.

If you execute CMS from a 3270 device, you must issue the ATTN or REQUEST command if you are in CP mode and want to return to the virtual machine mode.

The following two figures show the effect of an Attention interrupt when your virtual console is in either VM or CP mode. Figure 3 shows the action taken by VM/370 when your virtual console is in VM mode and you press the Attention key. The first column represents the condition of the terminal keyboard when you press the Attention key. The second column represents the number of times you press the Attention key. If your virtual console is in the VM mode, one Attention requests an interrupt and two quick Attentions force your virtual machine to the CP command environment. The third column represents the action VM/370 takes depending on the number of times you press the Attention key. "any" means that the effect of the Attention key is the same no matter how many times you press it.

Note: If the operator sets his terminal mode to VM and resets his virtual machine, his console acts as described in the first state of Figure 3 (that is, more than one attention is required to unlock the keyboard for CP input).

State of Terminal before Attention Key Pressed	No. of ATTN	Resulting Action
Terminal idle; keyboard locked; virtual machine running	1	Attention interrupt pending; virtual machine running
	>1	Keyboard unlocked for CP input
Terminal receiving output from virtual machine	1	Attention interrupt pending; virtual machine running
	>1	Keyboard unlocked for CP input at completion of console I/O
Keyboard unlocked for input to virtual machine; no data entered or all data deleted	1	Device end (DE) and attention status pending; virtual machine running[1]
	>1	Unit exception (UE) status pending; virtual machine running
Keyboard unlocked for input to virtual machine; some data entered	1	Unit exception (UE) status pending; virtual machine running
	>1	Device end (DE) status pending; keyboard unlocked for CP input
Keyboard locked; executing CP command	any	Attention ignored
Keyboard locked; in SLEEP mode entered via command	any	Keyboard unlocked for CP input
Keyboard locked; in SLEEP mode entered via Diagnose instruction	any	Virtual machine resumes execution
Terminal receiving output from CP but not from user command	1	Attention interrupt pending; virtual machine running
	>1	Keyboard unlocked for CP input
Terminal receiving output in response to CP command	any	Output line canceled and in some cases command output canceled
Keyboard unlocked for CP input; no data entered or all data canceled	any	Attention interrupt made pending; virtual machine running
Keyboard unlocked for CP input; some data entered	any	Input line canceled; keyboard unlocked for CP input

[1] To perform this function on a 3270, position the cursor one position to the left of the user input area (line 22, position 0) and press the Enter key.

Figure 3. Effects of Attention Interrupt While Virtual Console Is Set to the VM Terminal Mode

Figure 4 shows the action taken by VM/370 when your virtual console is in CP mode. If your virtual console is in CP mode and you press the Attention key one or more times, you force the virtual machine to the CP command environment.

State of Terminal before Attention key Pressed	Resulting Action
Terminal idle; keyboard locked; virtual machine running	Keyboard unlocked for CP input
Terminal receiving output from virtual machine	Keyboard unlocked for CP input
Keyboard unlocked for input to to virtual machine; no data entered or all data deleted	Unit exception (UE) status pending; keyboard unlocked for CP input
Keyboard unlocked for input to virtual machine; some data entered	Device end (DE) status pending; keyboard unlocked for CP input
Keyboard locked; executing CP command	Attention ignored
Keyboard locked; in SLEEP mode entered via command	Keyboard unlocked for CP input
Keyboard locked; in SLEEP mode entered via Diagnose instruction	Virtual machine resumes execution
Terminal receiving output from CP but not from user command	Keyboard unlocked for CP input
Terminal receiving output in response to CP command	Output line canceled and in some cases command output canceled
Keyboard unlocked for CP input; no data entered or all data canceled	Attention interrupt made pending; virtual machine running
Keyboard unlocked for CP input; some data entered	Input line canceled; keyboard unlocked for CP input

Figure 4. Effects of Attention Interrupt While Virtual Console Is Set to the CP Terminal Mode

This section describes your virtual disk system and how you use it. It describes how the disks are named, how they are accessed, when they need to be formatted, how to access and release them and, in general, how to manipulate them.

Your virtual machine can have many disks defined for it in the VM/370 directory, but it can have at most ten virtual disks logically accessed at any time. Virtual disks are logical subdivisions of real disks; but they can be considered real disks. Each has its own virtual device address, virtual cylinders, and for CMS disks, a Master File Directory, which lists the files contained on the disk.

You can request the operator to attach an entire real disk to your virtual machine. This real disk, once attached, is considered a virtual disk.

When your virtual machine is defined, the system programmer estimates your disk storage requirements and allocates cylinders for your virtual disks accordingly.

Virtual Disk Identifiers and Addresses

Virtual disks have disk identifiers and virtual addresses. The virtual disk identifier (or filemode letter) is a single-letter specification of A, B, C, D, E, F, G, S, Y, or Z. The disk identifier is a part of the filemode specification used in the CMS command line. The virtual disk address is a three-character hexadecimal number. The address can be assigned permanently in the VM/370 directory or temporarily via the CP DEFINE command.

THE A-DISK

The A-disk is the primary user disk. It is a read/write disk which is accessed when you begin your CMS terminal session. You can create, store, and modify files using this disk. Usually, the address of the A-disk is 191.

DISKS B THROUGH G, Y, AND Z

You may have several disks defined in your VM/370 directory entry. During a terminal session you may want to make these disks a part of your active virtual machine. In this case, you use the ACCESS command to acquire a disk and give it a name. The identifiers B through G, Y and Z are used to identify these disks.

If you have a virtual 192 disk defined for your virtual machine in the VM/370 directory, or if you specifically define a virtual disk with address 192 before loading CMS, that disk is accessed as your D-disk at the time your A-disk is accessed.

THE S-DISK

The S-disk is the system disk and is read-only. It contains the CMS nucleus and disk-resident command modules. The Y and Z disks can be extensions of the system disk, depending on your virtual machine configuration. If you have 190 and 19E disks defined in your VM/370 directory entry, these are accessed as the S and Y disks after you IPL CMS.

Formatting Virtual Disks

Before you use your virtual disks, you must ensure that they have been formatted. Each disk must be formatted the first time you use it; it need not be formatted thereafter.

Use the CMS FORMAT command to format a virtual disk the first time you use it. However, the FORMAT command erases the contents of the disk, so you should take care not to issue FORMAT for a disk whose contents you want to save.

Each time you use the CP DEFINE command to attach a temporary virtual disk to your virtual machine, you must issue the FORMAT command.

If you are formatting a disk to contain OS or DOS files, use the IBCDASDI virtual disk initialization program, which is described in the VM/370: Operator's Guide.

Virtual Disk Addresses and How They Are Defined

You can have both permanent and temporary disks attached to your machine during a terminal session. Permanent disks are predefined in the VM/370 directory entry for your virtual machine. Temporary disks are those you define for your own virtual machine using the CP DEFINE command.

PERMANENT VIRTUAL DISKS

The VM/370 directory entry for your virtual machine defines the permanent virtual disks. Each disk has associated with it an access mode specifying whether you can read or write on the disk or only read from it. The virtual disk entries in the VM/370 directory may typically look like the following:

```
MDISK  197  2314  000  050  CMS190  R
MDISK  198  3330  010  005  CMS001  W
MDISK  194  3330  050  005  CMS192  W
```

The first two fields describe the device, virtual disk in this example, and the virtual address of the device. The third field specifies the device type of your virtual disk. The fourth and fifth fields specify the starting real cylinder at which your virtual disk logically begins and the number of cylinders allocated to your virtual disk, respectively. The sixth field is the label of the real disk on

which the virtual disk is defined and the seventh field is a letter specifying the read/write mode of the disk. The MDISK control statement of the Directory Service Program is described in the VM/370: Operator's Guide.

Even though these devices are permanently attached to your virtual machine, you must access them using the CMS ACCESS command before you can use them.

DEFINING TEMPORARY VIRTUAL DISKS

Using the CP DEFINE command, you can attach a temporary disk to your virtual machine for the duration of a terminal session. Once attached, the disk must be accessed using the CMS ACCESS command. For example, the commands below allow you to use a temporary G-disk.

```
define t3330 as 291 cyl 10
```

You should then format it using the CMS FORMAT command:

```
format 291 g
```

You should respond to the CMS response messages.

Accessing and Releasing Virtual Disks

Even though virtual disks may be defined in the VM/370 directory entry for your virtual machine, you cannot use them until you access them. There may be many disks defined in that entry, but your CMS virtual machine may have access to only ten of them at a time. The CMS ACCESS command allows you to logically access a virtual disk for your CMS virtual machine.

Since you can have only ten virtual disks active in your CMS machine at a time, you may want to release a disk so that you can access another. You use the CMS RELEASE command to logically release the virtual disk from your CMS virtual machine, then access the new disk using the ACCESS command.

Linking to Another User's Virtual Disk

VM/370 lets you link to a virtual disk owned by another user. Use the CP LINK command as shown in the example below:

```
link to bensid 193 as 194 r pass= go
```

This command links another user's virtual disk (whose address is 193) to your virtual machine, where it has the address 194. The R specifies that the disk can only be read and GO is the read password. BENSID is the userid for the virtual machine to which you are linking.

You may also have another user's virtual disk defined in your VM/370 directory entry.

Once you have linked a disk to your virtual machine, use the CMS ACCESS command to access it and give it a name.

Extending One Virtual Disk from Another

Using the CMS ACCESS command, you can make one virtual disk a read-only extension of another so that when you specify that a disk is to be read, any extension to that disk is read also. For example, the command:

```
access 192 d/a
```

makes the D-disk an extension of the A-disk.

When one virtual disk is an extension of another, the extension disk is, by definition, a read-only disk. However, you can respecify its access status by issuing the ACCESS again, this time not extending the D-disk from the A-disk.

```
access 192 d
```

Not only does extending one disk from another alter its access status, it may also affect the search order for the disk accessed on your system.

When you issue a CMS command and specify * as the filemode, the extensions set via the ACCESS command are not in effect, that is, the search is in alphabetical order.

Only one level of disk extension is permitted.

Virtual Disk Search Order

When you specify a file to be acted upon in a CMS command, the default (or standard) CMS search for that file is in alphabetical order; the A-disk first, B second, and so on. The standard search order is used when no mode letter is specified or implied. If a mode letter is specified or implied, the search order is first the given disk and then all the disks that are read-only extensions of the given disk (except for certain commands, such as LISTFILE and TAPE DUMP, which purposely ignore read-only extensions).

Read/Write Status of Virtual Disks: R/O and R/W

The read/write status for a virtual disk defines whether you can read or read and write on a disk on your virtual machine. You can access disks in two ways: read-only, where files on that disk can only be read; and read/write, where files can be read and written.

To access a disk, you must:

1. Identify a disk as part of your virtual machine configuration. If the disk appears in your VM/370 directory entry it is already a part of your virtual machine configuration. Also, you can make a disk part of your configuration by issuing a CP LINK or DEFINE command.

2. Identify the disk to CMS and assign it a file directory name. You issue the ACCESS command after you load CMS to do this. The CMS ACCESS command associates a particular disk with a given file directory name and, optionally, specifies which files on the disk are to be used and specifies the disk as read-only.

The following example shows how you add a temporary disk and a user disk to your CMS virtual machine.

```
ipl cms
link dept637 230 197 r 12601
define t3330 as 192  cyl 5
format 192 d
access 197 b
```

First the LINK command adds a device at virtual address 197 to your virtual machine. (The disk added is defined in the VM/370 directory for a virtual machine with a userid of DEPT637 as device address 230 with a read password of 12601.)

Then the DEFINE command adds temporary disk space (from a CP pool of such space) to your virtual machine at address 192.

The CMS FORMAT command initializes the temporary disk area (192) in the CMS format.

The ACCESS command activates the disk at virtual address 197 (similar to VARY ONLINE in OS) and assigns the disk the file directory name B. The importance of the directory name was explained in the preceding section "Virtual Disk Search Order"

If ACCESS is not the first command entered after CMS is loaded, an automatic ACCESS is performed to access a disk at device address 191 as the A-disk.

Both CP and CMS can control read/write access to disks, as is illustrated in Figure 5.

	CMS ACCESS	
CP ACCESS	Read-only	Read/write
Read-only	Read-only	Read-only
Read/Write	Read-only	Read/Write

Figure 5. CP and CMS Disk Access

Access allowed by CP is determined by the VM/370 directory entry or the form of LINK issued by a virtual machine operator for a particular disk.

The read/write status of virtual disks can be controlled in several ways:

- It can be specified explicitly in the VM/370 directory.
- It can be specified by the CP DEFINE command.
- A disk can be made read-only by extending it from another disk.

If the read/write status is defined in the VM/370 directory, it is either a read/write (R/W) or read-only (R/O) disk. R/W disks can be read from and written on; R/O disks only may be read.

When you define a temporary disk at your terminal, that disk is always a R/W disk.

The read/write status of a particular disk affects the way you use it. R/O disks cannot be written on. For example, you cannot use a R/O disk to contain the output of a language processor.

You can find the read/write status of a file by issuing the QUERY command with the DISK option.

The file is the essential unit of data in the CMS system. A CMS file is a logically grouped unit of data you define via a CMS command. This section gives you information on how you can create a file and name it, how files are used in the system, and how CMS handles OS data sets and DOS files.

Creating or Defining Files

You can create a file using the CMS Editor, which is invoked with the CMS EDIT command. The Editor lets you create and modify a file on your virtual disk. The FILEDEF command defines data sets created under an OS system such as OS/VS1 or OS/VS2 so that they can be recognized by CMS. You use OS macros to perform I/O operations for these data sets.

The files (or data sets) defined by FILEDEF can be OS data sets created by an OS system, DOS files created by a DOS system, or files created under CMS using OS macros simulated by CMS and residing on a CMS virtual disk. You can then access them under CMS just as you do files you create using the Editor. Files may also be introduced to the system by means of the READCARD, DISK, and TAPE commands.

All the information you need to create a CMS file using the Editor can be found in the VM/370: EDIT Guide. The FILEDEF command is described later in this section and in Section 7.

In order to create a file, you must define a name for the file and decide upon the filetype of the file; use of the file determines its filetype. This information is entered along with the command in the form of a file identifier. Appendix F lists the reserved filetypes and describes the use of each.

For a description of how to load existing source card decks into your virtual machine, see the CMS READCARD command description.

Naming Your Files: The File Identifier

When you create a file in CMS, you name it using a file identifier. The file identifier consists of three fields: the filename (fn), the filetype (ft), and the filemode (fm). This file identifier is then used to refer to the file while you are using CMS.

THE FILENAME FIELD

The filename for a file is a one- to eight-character alphameric symbol. The characters can be A through Z, a through z, 0 through 9, and the special characters $, #, or @. You should be careful when using the # and @ characters, since they are used also as VM/370 logical editing characters.

You can use any filename you wish: however, you should avoid duplicating names or abbreviations for CP or CMS system commands. If you duplicate the name of a system command and invoke the name from CMS, the execution of that file depends on the CMS command search order, which is described in Section 2 under "CMS Command Search Order."

Filenames, in some cases, become user-defined command names. The filenames for files with the filetype MODULE or EXEC have the same effect as a command, in many cases.

THE FILETYPE FIELD

The filetype is also a one- to eight-character alphameric symbol. The characters can be A through Z, a through z, 0 through 9, and the special characters $, #, or @. You can find more information about the filetype specification in the sections that follow.

Certain filetypes have special meaning to CMS, that is, CMS assumes the file has specific characteristics such as record length, tab settings, truncation column, upper or lowercase, and other information of significance for use by a command. These special filetypes are described in a later section "CMS Filetypes" and are summarized in Appendix F.

THE FILEMODE FIELD

The filemode field has two characters: the filemode letter and the filemode number. The filemode letter is established by the ACCESS command, and specifies the virtual disk on which a file resides: A through G, S, Y, or Z. The filemode number can be a number from 0-5. Figure 6 shows what each filemode number means. The first column is the filemode number, the second column specifies the read/write status associated with the number, and the third column gives a brief description of how to use the filemode number.

Filemode Number	Read/Write Status	Meaning
0	R/W	The file specified is a private file; you cannot access a file with the 0 filemode unless you have read/write privileges for the virtual disk on which the file resides.
1	R/W	You can read from and write into this file, depending on how the disk is accessed.
2	R/W	You can read from and write into this file, depending on how the disk is accessed. Certain files on the S-disk are mode 2; you can access these files. You can also use mode 2 to describe files on disks other than the system disk.
3	R/E	The file is to be erased after it is read. Usually, this filemode is used for temporary work files created by the language processors and some CMS commands.
4	OS	This file is created using OS macros. It may be blocked and, if in OS variable format, may contain Block Descriptor Words (BDWs) and Record Descriptor Words (RDWs).
5	R/W	Has the same meaning as filemode 1.
6-9		Reserved for IBM use.

Figure 6. Determining Filemode Numbers

The other information you need when using this field of the file identifier is found later in this section, "How to Specify the Filemode Field."

CMS Filetypes

The filetype field specifies the type of the file; that is, how the file functions in the CMS system. The filetype field is used by the EDIT command to define standards for record length, tab settings, upper or lowercase, truncation column, and so on. For other commands, the filetype field is used to identify a subset of all files that are appropriate for processing by that command, for example you may want to process all files with the filetype ASSEMBLE.

Source files have a filetype describing the language in which the file is written; files containing relocatable object code have the filetype TEXT; files containing executable object code have the filetype MODULE; files containing listings have the filetype LISTING, and so on.

The filetypes described in this section are reserved; that is, they have a special meaning to CMS when you specify them in the filetype field of the file identifier. They are most useful in the programming environment (for example, filetypes describing source files, EXEC, TEXT, MODULE, and LISTING files). There is also a description of files created during certain types of processing under CMS.

FILETYPES FOR ASSEMBLER AND COMPILER SOURCE FILES

When you create a source file using the Editor you assign a filetype that describes the source language used to create the file. For example, when you create an assembler language file using the CMS Editor, the command you issue might be:

```
edit myfile assemble
```

where MYFILE is the filename of the file and ASSEMBLE is its filetype. When you specify the filetype as ASSEMBLE, you are telling the Editor that the file is an assembler language file, and that the records created should be appropriate for processing by the system assembler. The Editor uses the filetype ASSEMBLE to define suitable conditions for creation of an assembler language source file. That is, the Editor creates a file consisting of fixed-length, 80-character records, with all input translated to uppercase, and with standard tab settings of 1, 10, 16, 31, 36, and so on.

See "Appendix F. Reserved Filetype Descriptions" for a list of the filetypes associated with the other supported programming languages and the CMS commands. When you create source files, there are rules concerning usage of the file, the command you use to process the file under CMS, the record format of the file and the contents of the file.

OBJECT FILES: FILETYPE TEXT

When you execute one of the language processors, the results of the processing (assembled or compiled object code) are placed in a file with the same filename as the source file for the program, but with a filetype of TEXT.

FILES WITH THE FILETYPE LISTING

Also, when you execute the language processors, a listing describing the source code and the results of execution is contained in a file with the same filename as the source file and filetype LISTING.

FILES WITH THE FILETYPES EXEC AND MODULE

There are two types of files that can be executed by invoking the filename of the file. These two types of files are files that have either the filetype EXEC or the filetype MODULE.

An EXEC file may be a procedure (a sequence of predefined commands to be executed as a unit). EXEC procedures also provide you with a conditional execution capability similar to the looping facilities of high level languages. You can find information about how to write and use EXEC procedures in the VM/370: EXEC User's Guide.

In CMS, a MODULE file is a non-relocatable copy of a program or routine that resides in storage in executable form. A MODULE file is created by loading an object file (filetype TEXT) in your virtual machine (via the LOAD command) and then generating the module via the GENMOD command. Like the EXEC file, the MODULE file can be executed by invoking its filename.

FILE GROUPS CREATED BY THE LANGUAGE PROCESSORS

Many CMS commands create groupings of files, each related to the other by the same filename. Some of these files are permanent and some are temporary. For example, if you issue the command

```
assemble myfile
```

the system assembler executes to assemble the file named MYFILE. Execution of the assembler generates several files, some permanent and some temporary. You can request CMS to list the permanent files by means of the LISTFILE command:

```
listfile myfile * a1
```

CMS then generates a list of all files with the filename of MYFILE, including the permanent files created during the execution of the assembler, which might look like this:

```
MYFILE ASSEMBLE A1
MYFILE TEXT     A1
MYFILE LISTING  A1
```

where the TEXT file contains the object code resulting from the assembly, and the LISTING file contains the program listing generated by the assembly. The source input file, MYFILE ASSEMBLE A1, is not erased.

Temporary files are also created by the assembler for use as workfiles:

```
MYFILE SYSUT1
MYFILE SYSUT2
MYFILE SYSUT3
```

The filetypes of the workfiles may vary depending on the processor you are using. Disk space is allocated for the assembler work areas on an as-needed basis. They are erased when processing is complete.

You should ensure that a file created during the execution of a language processor does not have the same file identifier as one you wish to save. CMS, when instructed by a language processor to write a file, erases any file with the same file identifier as the one specified.

Using OS Programs and Macros under CMS

You can assemble and execute programs under CMS that require the use of OS macros simulated by CMS. Figure 7 lists the OS macros that CMS simulates.

Macro	SVC Number	Function
XDAP	00	Read or write direct access volumes
WAIT	01	Wait for an I/O completion
POST	02	Post the I/O completion
GETMAIN	04	Conditionally acquire user storage
FREEMAIN	05	Release user-acquired storage
GETPOOL	–	Simulate as SVC 10
FREEPOOL	–	Simulate as SVC 10
LINK	06	Link control to another phase
XCTL	07	Delete, then link control to another load phase
LOAD	08	Read a phase into storage
DELETE	09	Delete a loaded phase
GETMAIN/ FREEMAIN	10	Manipulate user free storage
TIME	11	Get the time of day
ABEND	13	Terminate processing
SPIE	14	Allow processing program to handle program interrupts
BLDL/FIND	18	Manipulate simulated partitioned data files
OPEN	19	Activate a data file
CLOSE	20	Deactivate a data file
STOW	21	Manipulate partitioned directories
OPENJ	22	Activate a data file
TCLOSE	23	Temporarily deactivate a data file
DEVTYPE	24	Obtain device-type physical characteristics
TRKBAL	25	NOP
WTO/WTOR	35	Communicate with the terminal
EXTRACT	40	Effective NOP
IDENTIFY	41	Add entry to loader table
ATTACH	42	Effective LINK
CHAP	44	Effective NOP
TTIMER	46	Access or cancel timer
STIMER	47	Set timer
DEQ	48	Effective NOP
SNAP	51	Dump specified areas of storage
ENQ	56	Effective NOP
FREEDBUF	57	Release a free storage buffer
STAE	60	Allow processing program to decipher ABEND conditions
DETACH	62	Effective NOP
CHKPT	63	Effective NOP
RDJFCB	64	Obtain information from FILEDEF command
SYNAD	68	Handle data set error conditions
BSP	69	Backup a record on a tape or disk
GET/PUT	–	Access system-blocked data
READ/WRITE	–	Access system-record data
NOTE/POINT	–	Manage data set positioning
CHECK	–	Verify READ/WRITE completion
TGET/TPUT	93	Read or write a terminal line
TCLEARQ	94	Clear terminal input queue
STAX	96	Create an attention exit block
RETURN	–	Return from a linked or attached routine

Figure 7. OS Macros Simulated by CMS

ASSEMBLING A PROGRAM USING OS MACROS

To assemble a program that uses OS macros, you must first issue a GLOBAL command to make the macro library containing the macros available to the assembler.

```
global maclib  osmacro
```

Once you have accessed the macro library, you can use macros from the library to assemble a program like the one shown in the following example:

```
TESTER  CSECT
          .
          .
          .
        OPEN  (OUTDCB,(OUTPUT))
          .
          .
          .
        PUT   OUTDCB,AREA
          .
          .
          .
        CLOSE  OUTDCB
          .
          .
          .
OUTDCB  DCB   DDNAME=OUT, ...
```

To assemble the example, issue the command:

```
assemble tester
```

After the file is assembled, you can load the resulting TEXT file and then execute the program.

EXECUTING A PROGRAM THAT USES OS MACROS

In order to execute a program that uses OS macros, you must associate each DCB statement in your program with the device specified to perform the input/output operation for it. As in OS, this association is made via the ddname. In CMS, however, you issue the FILEDEF command, which performs a function for CMS that parallels the function performed by the DD statement in OS.

Thus, to execute the preceding example program, you issue these commands:

```
filedef out disk tester output a1
load tester
start
```

When you use the OS macros simulated by CMS to read and write (as in the example), you can write only to CMS disks. You cannot issue a write to an OS or DOS disk. Also, CMS does not simulate all reads and writes; only the OS BSAM, BPAM, QSAM, and BDAM access methods are simulated.

Files written on a CMS disk can, in turn, be read by programs running under CMS when those programs request that CMS simulate a read (for instance a GET or READ macro).

READING OS DATA SETS FROM OS DISKS AND DOS FILES FROM DOS DISKS

Sequential and partitioned data sets residing on OS disks, and sequential files residing on DOS disks, can be read by programs using OS macros running under CMS. Also, certain CMS commands can be used to process these data sets and files on OS and DOS disks. Figure 8 lists commands you can use to manipulate OS data sets and DOS files under CMS, and briefly describes the function of the command in relation to its use with OS data sets and DOS files.

Command	Operation
ACCESS	Makes the OS or DOS disk containing the OS data set or DOS file available in R/O status to your CMS virtual machine.
RELEASE	Releases the OS or DOS disk you have accessed (via ACCESS) from your CMS virtual machine.
LISTDS	Lists information describing OS data sets residing on an OS disk, or DOS files on a DOS disk.
STATE	Verifies the existence of an OS data set or DOS file on a disk. Before STATE can verify the existence of the data set or file, you must have defined it (via FILEDEF).
FILEDEF	Defines the OS data set or DOS file for use under CMS by associating an OS ddname with an OS data set name or DOS file-id. Once defined by FILEDEF, the OS data set or DOS file can be used by an OS program running under CMS and can be manipulated by the other commands that support OS functions.
DDR	Copies an entire OS or DOS disk to tape.
GLOBAL	Makes macro libraries available to the assembler. You can prepare an OS macro library for reference by the GLOBAL command by issuing FILEDEF for the data set and giving the data set a filetype of MACLIB. Note: If you are going to assemble, remember that the ddname to use on the FILEDEF command must be CMSLIB.
ASSEMBLE	Assembles an OS data set or DOS file under CMS.
MOVEFILE	Moves data records from one device to another device. Each device is specified by a ddname, which must have been defined via FILEDEF.
QUERY	Lists (1) the status of virtual machine features specified by the CMS SET command, (2) the files that have been defined via FILEDEF in your system of virtual disks, and (3) the status of virtual disks attached to your virtual machine.

Figure 8. CMS Commands Used in Processing Data Sets on OS Disks and Files on DOS Disks

Restrictions for Reading OS Data Sets

The following restrictions apply when you read OS data sets under CMS:

- Read password-protected data sets are not read.
- VSAM, BDAM, and ISAM data sets are not read.
- Multivolume data sets are read as single-volume data sets. End-of-volume is treated as end-of-file and there is no end-of-volume switching.
- Keys in data sets with keys are ignored; only the data is read.
- User labels in user-labeled data sets are bypassed.

Restrictions for Reading DOS Files

The following restrictions apply when you read DOS files under CMS:

- No DOS macros are simulated.
- Only DOS sequential files can be read. CMS operands and options that do not apply to OS sequential data sets (such as the MEMBER and CONCAT options of FILEDEF and the PDS option of MOVEFILE) also do not apply to DOS sequential files.
- The following types of DOS files cannot be read:

  -DOS VSAM, DAM and ISAM files.

  -DOS core image, relocatable, source statement and procedure libraries.

  -Files with the input security indicator on.

  -Files that contain more than 16 user label and/or data extents. (If the file has user labels, they occupy the first extent; therefore the file must contain no more than 15 data extents.)

- Multivolume files are read as single-volume files. End-of-volume is treated as end-of-file. There is no end-of-volume switching.
- User labels in user-labeled files are bypassed.
- Since DOS files do not contain BLKSIZE, RECFM, or LRECL options, these options must be specified via the FILEDEF command or the DCB statement; otherwise, defaults of BLOCKSIZE=32760 and RECFM=U are assigned. LRECL is not used for RECFM=U files.
- If a DOS file-id does not follow OS naming conventions (that is, one- to eight-byte qualifiers with each qualifier separated by a period; up to 44 characters including periods), you must use the DSN ? operand of FILEDEF and the ? operand of LISTDS to enter the DOS file-id.

Using the FILEDEF and MOVEFILE Commands

The following examples show how to use the FILEDEF and MOVEFILE commands to handle OS data sets and DOS files under CMS.

The following sequence of CMS commands moves an OS STOW macro file from an OS partitioned data set called SYS1.MACLIB or a CMS file called SYS1 MACLIB to the CMS file STOW MACRO.

```
access 195 b/a
filedef test1 disk sys1 maclib b1 (member stow)
filedef macro disk stow macro
movefile test1 macro
```

The following sequence of CMS commands moves all the members of an OS partitioned data set called SYS1.MACLIB or a CMS file called SYS1 MACLIB into separate CMS files, each with a filename equal to its member name and a filetype of MACRO.

```
access 195 b
filedef test2 disk sys1 maclib b1
filedef macro disk
movefile test2 macro (pds)
```

Either of the following sequences of CMS commands can be used to verify the existence of the OS data set called TEST.OS.SAMPLE.1 and assemble it with SYS1.MACLIB on an OS disk and CMSLIB on a CMS disk.

```
access 198 d
listds test os sample 1 *
filedef assemble d1 dsn test os sample 1
filedef cmslib disk sys1 maclib * (concat
global maclib sys1 cmslib
assemble file

  -- or --

access 198 d
filedef assemble disk test1 assemble d1 dsn ?
test.os.sample.1
state test1 assemble *
filedef cmslib disk cmslib maclib * (block 3360 lrecl 80 recfm fb concat)
filedef cmslib disk sys1 maclib * (concat)
global maclib cmslib sys1
assemble test1
```

The following sequence of CMS commands moves a DOS file named DAILY ACCOUNT JAN 1 from the DOS 195 disk to the CMS file FILE OUT.

```
access 195 b
filedef in disk file in b1 dsn ?
DMSFLD220R ENTER DATA SET NAME
daily account jan 1
movefile in out
(A default FILEDEF is issued for FILE OUT.)
```

USING PROGRAM PRODUCTS UNDER CMS

The CMS assembler and the Program Product language processors supported by CMS (listed in the VM/370: Introduction) are OS programs that require definition via FILEDEF in order to perform input/output processing. For input, all of these programs can use sequential or partitioned data sets that reside on OS disks, or sequential files that reside on DOS disks.

When a supported language processor is executed under CMS, CMS issues FILEDEFs for the data sets or files it requires in order to execute the program. You can override these default definitions by issuing a FILEDEF for those data sets or files before you invoke the language processor. You must issue the FILEDEF for the data set or file each time you want to use it, since the processors clear all file definitions when they complete their processing.

When OS compilers execute under CMS, they normally:

- Run the compilation to completion.
- Display any diagnostics at the terminal.
- Generate a CMS disk file with the same filename as the source program and a filetype of TEXT, which contains the object deck created by the assembler or compiler.
- Direct the printed output of the assembler or compiler to the spooled printer or to a disk file with a filetype of LISTING.

The filename of files created by the assembler or compilers running under CMS is equal to that of the source file.

Disk work files required by the assembler or OS compilers under CMS are automatically created during compilation and erased at the end of compilation. No cataloging or erasing of data sets is required.

Object programs (TEXT files) produced under CMS and under OS in real or virtual machines may be executed under CMS if they do not utilize certain OS functions not simulated by CMS. (OS macro functions that are simulated are discussed in "Using OS Programs and Macros under CMS.") Object programs (except for the PL/I Checker) using non-simulated OS macro functions must be transferred to an appropriate real or virtual OS machine for execution. PL/I Checker programs that use non-simulated OS macro functions must be both assembled and executed on the appropriate real or virtual OS machine.

How to Specify the Filemode Field

The filemode field of the file identifier is comprised of two characters: a virtual disk letter and a filemode number. In almost all CMS commands, this field is optional. You need not specify it since CMS itself has the ability to search for the file. Some commands, in fact, do not permit a filemode entry.

There are four ways to specify this field of the file identifier:

- Explicitly, by actually entering the filemode letter and filemode number in the filemode field.
- Implicitly, by leaving the entire field blank.

- By specifying an asterisk (*) in the filemode field.
- By specifying an equal sign (=) in the filemode field.

If you specify the filemode explicitly, the virtual disk you specify and any extensions of it are searched for your file. If, however, the virtual disk you specify is an extension, the extension disk is searched, but not the parent disk. If you specify a read-only virtual disk when you are writing a file, CMS returns an error message.

If you specify the filemode field implicitly by leaving it blank, only your A disk and its extensions are searched for the file.

The asterisk (*) can have two meanings, depending on the command with which it is specified: (1) search all disks until the specified file is found, or (2) search all disks for all occurrences of the file.

The equal sign (=) as the filemode is specified when you want to read from and write on the same virtual disk. This character can be used only on disks for which you have read/write privileges.

When you are using one of the Program Product language processors to write to a virtual disk, CMS first attempts to write to the virtual disk you specify and then attempts to write to that disk's parent (if one exists). If there is no room on either of these disks, or if both disks are read-only, writing is done only to the A-disk.

SPECIFYING SEARCH ORDER USING THE FILEMODE FIELD

A CMS function that is trying to locate a file on a disk may search for that file in either of two ways:

- If it is searching for a file with a particular filemode, such as "A", that disk and its immediate extensions are searched.
- If it is to search all accessed disks (Filemode *), the disks are searched in alphabetical order (A B C D E F G S Y Z).

CMS functions or commands that do not allow you to specify the filemode (such as LOAD, ASSEMBLE, and MACLIB) use the second type of search, and search all accessed disks in alphabetical order.

Commands that allow you to specify the filemode may do either type of search. If you specify a particular disk or allow it to default to "A", then the first type of search is done. If you specify filemode as "*", then the second type of search is done.

However, the LISTFILE and TAPE DUMP commands do not use extensions in search of a particular disk.

Libraries

CMS provides two types of libraries: macro libraries and text, or program libraries. A library is a file with the filetype MACLIB or TXTLIB. Unlike other CMS files, a library file consists of members plus a dictionary for locating the members by name within the library file.

A macro library is a file whose filetype is MACLIB. It contains a dictionary and members which are macro definitions. The system macro libraries are CMSLIB MACLIB, OSMACRO MACLIB, OSMACRO1 MACLIB and TSOMAC MACLIB. CMSLIB MACLIB contains the CMS macros. OSMACRO MACLIB contains selected OS macros from the OS macro library, SYS1.MACLIB, which are

simulated by CMS. OSMACRO1 MACLIB contains the other macros from SYS1.MACLIB, making it possible to assemble a program in CMS for OS execution. And, TSOMAC MACLIB contains selected TSO macros.

You can create your own macro library by using the MACLIB command. Files with the filetype MACRO or COPY can be specified in the MACLIB command for inclusion in the library. The MACLIB command can also be used to add, delete, or replace macros in an existing library; to list the name, size, and location of macros in a library; and to compress a library.

A text library is a file whose filetype is TXTLIB. It contains a dictionary and members that are relocatable text files. The system text library CMSLIB TXTLIB contains extended precision floating point simulation routines. If you are a TSO user, you can also access the TSOLIB TXTLIB.

You can generate your own text library by using the TXTLIB command. Only files with the filetype TEXT may be included. A maximum of 1000 control section names and entry points can be contained in a TXTLIB file. The TXTLIB command may also be used to add or delete members from a library or to list the entry points, control section names, and the location of members of the library.

CMS has no automatic library calls. A library must be made available using the GLOBAL command before CMS will search it. GLOBAL has two forms, the TXTLIB form for specifying text libraries, and the MACLIB form for specifying macro libraries. The command GLOBAL MACLIB CMSLIB OSMACRO would make CMSLIB MACLIB and OSMACRO MACLIB available to be searched during an assembly for macro operation codes, and they would be searched in the order they were named in the GLOBAL command. Text libraries that are made available with the GLOBAL command are searched for missing subroutines or undefined filenames whenever the LOAD or INCLUDE commands are issued. Up to eight libraries may be specified with either form of GLOBAL. The commands GLOBAL TXTLIB and GLOBAL MACLIB, specifying no library names, terminate searching of previously specified libraries.

Descriptions of the MACLIB, TXTLIB, and GLOBAL commands appear in Section 7.

CMS Tape Handling

CMS has two commands that handle tapes.

The CMS TAPE command dumps CMS formatted files from disk to tape, loads such files from tape to disk, and performs various control operations on a specific tape, such as setting tape modes, forward or backward spacing, and rewinding the tape. The TAPE command is used solely for CMS files. The files on tape are created in a unique format that can be read only by the TAPE LOAD command.

The TAPPDS command reads tapes created by the OS utility programs such as IEHMOVE, IEBUPDTE, and IEBPTPCH. If the tape contains an OS partitioned data set (PDS) produced by the OS IEBPTPCH utility, the TAPPDS command can write the members as individual CMS files. Optionally, it produces CMS files from tape in the OS IEBUPDTE control file format, blocked or unblocked. TAPPDS can also create CMS files from unloaded partitioned data sets in IEHMOVE format, blocked or unblocked. The tape may be unlabeled or may have a standard OS label.

| Note: The MOVEFILE and DDR commands, as well as user-written programs, can handle tapes.

CMS Unit Record Support

CMS supports one virtual card reader at virtual address 00C, one virtual card punch at virtual address 00D, and one virtual printer at virtual address 00E. Under VM/370, these devices are spooled. CMS does not support real or dedicated unit record devices, nor does it support a virtual 2520 Card Punch.

CARD READER

The READCARD command reads data records from the spooled card reader to a CMS disk. Input records of 151 or less characters are accepted. Column binary data is not acceptable. All user-generated card decks must be read into the virtual reader before a READCARD command can be issued. You can get card decks into a CMS reader in either of two ways:

- A card deck, containing only one file, is placed in a real card reader and read by CP. The deck must be preceded by a VM/370 identification (ID) card specifying the userid of the virtual machine to receive the card images. (Files must be read in separately, even if they are for the same virtual machine. If a second deck is placed behind the first, the second ID is ignored and the second file is treated as though it were part of the first file. The only end-of-file condition recognized is an end-of-file on the real device.) The card images are placed on a spool file in the specified virtual machine's virtual card reader. If the specified user is not logged on to the system, the deck remains in his virtual card reader until he logs on and issues the READCARD command.

- One virtual machine transfers records from its virtual card punch or printer to a virtual card reader (its own or that of another user).

Once you have read the preceding four sections, you can use the sample program in this section as an introduction to the interactive facilities of VM/370. This section contains the information you need to create and use a program that runs under the CMS operating system. The text is organized so that you can enter the sample program at your terminal as you read.

The sample program treats a number of VM/370 commands in a "one-situation" manner. That is, most of the commands can be used in ways other than that shown in the sample program.

The first part of this section describes the steps you take before creating the program: learning to use your terminal, contacting the computer, loading the CMS operating system and formatting a disk with which to work.

The second part of this section deals with two topics: how to use the CMS Editor to create a program and how to use VM/370 commands to execute it.

The program you write is a simple one, even though it is coded in assembler language. It consists of only a few statements; all you need to do is enter them. When the program executes, you can use it interactively, so you get an idea of how to use VM/370 interactively.

Getting Started

VM/370 is an operating system that provides you with a unique facility--your own machine. This machine is a virtual machine, that is, a machine with software counterparts for almost all of the hardware components of a real machine. All of these components are controlled from your terminal, which is your virtual console.

On the real machine console, there are lights and keys that help you manipulate the machine; on the virtual console (your terminal), you manipulate the machine using the command languages of the CP and CMS operating systems.

The CP command language lets you manipulate your virtual machine components. For example, you could temporarily define a new device for your virtual machine using the CP DEFINE command. The CMS command language allows you to manipulate the virtual disks on your system and the files contained in them. For example, you can create and assemble a program file in assembler language using the CMS EDIT and ASSEMBLE commands.

WHAT YOU SHOULD KNOW BEFORE YOU CAN USE CP AND CMS

Before you can use CP and CMS, you should know (1) how to operate your terminal and (2) your userid (user identification) and password.

The Terminal: Your Virtual Console

There are many types of terminals you can use as a VM/370 virtual console. Before you can conveniently use any of the commands and facilities described in this section, you have to familiarize yourself with the terminal you will be using. Generally, you can find information about the type of terminal you are using and how to use it with VM/370 in the VM/370: Terminal User's Guide. If your terminal is a 3767, you also need the IBM 3767 Operator's Guide. The sample program is designed for typewriter terminals. Where parallel commands are needed for display terminals, you can find these commands described in the VM/370: EDIT Guide.

Your Userid and Password: Keys into the System

Your userid is a symbol that identifies your virtual machine to VM/370 and allows you to access VM/370. Your password is a symbol that functions as a protective device ensuring that only those authorized to use your virtual machine can use it. Both symbols are usually defined by the system programmer for your installation and you can obtain them from him.

When you are familiar with your terminal and know your userid and password, you are nearly ready to use VM/370. But before you begin using your terminal to enter commands, you should know how the VM/370 Logical Editing Characters can help you correct the typing mistakes you may make at your terminal.

Correcting Data at Your Virtual Console

VM/370 has a set of symbols you can use to correct typing errors and to change data as you enter lines at your keyboard. Using these logical editing symbols, you can cancel a line entirely, change a character in a line, logically end a line and begin a new one without pressing the Enter Key (or equivalent) and ignore the special meaning of a character on a line. Figure 9 lists the default logical editing symbols. To change the symbols' use, you must either define different logical editing symbols in your VM/370 directory entry or issue the CP TERMINAL command.

Symbol	Function
@	Logical Character Delete
#	Logical Line End
¢	Logical Line Delete
"	Logical Escape

Figure 9. Logical Editing Symbols

When you enter your input lines using these symbols, you do not see the effect they have immediately, since VM/370 is simply accepting the input you are entering. You can see the effect the logical editing symbols have when you request VM/370 to show you your input file.

The Logical Character Delete Symbol (@): Deletes the character preceding it on the input line. A string of character delete symbols deletes a corresponding number of preceding characters. For example, if you make a typing error such as:

```
permnaent
```

just enter five character delete symbols and follow them with the correct letters, as follows:

```
permnaent@@@@@anent
```

when you press the Enter Key on your terminal, the line is accepted by VM/370

```
permanent
```

The Logical Line Delete Symbol (¢): Causes your virtual machine to delete the logical line you are entering. For example,

```
a formula of the highest affirmation ¢
```

causes the entire line to be ignored by VM/370. You can press the Enter key to continue typing, or you can continue on the same input.

The Logical Line End Symbol (#): Causes your virtual machine to logically end the line you are entering. This symbol allows you to enter many logical lines on one physical line, for example,

```
a formula#of the highest#affirmation
```

is accepted by VM/370 as

```
a formula
of the highest
affirmation
```

The Logical Escape Symbol ("): Tells your virtual machine to ignore the special line-editing meaning of the character that follows. You use the logical escape symbol to ignore the line-editing meaning of the special characters that you do not want VM/370 to misinterpret. For example, VM/370 misinterprets the following line:

```
1 gross #2 pencils @ 92¢ per dozen
```

The special characters on this line are interpreted by VM/370 as:

- Begin a new line after "gross"
- Delete the blank space after "pencils"
- Erase everything starting with the # sign through the ¢.

However, the line is correctly interpreted if entered as follows:

```
1 gross "#2 pencils "@ 92"¢ per dozen
```

Note: The user may not be able to predict the results of mixing three or more consecutive line editing symbols because of the way the input line is acted upon by VM/370. For example:

```
abc""@def
```

and

```
abc""@@def
```

both result in

```
abcdef
```

CONTACTING VM/370 AND LOGGING ON

The next steps to take before you begin to create the program are: (1) contacting VM/370 and (2) identifying yourself to the computer.

Contacting VM/370

To contact VM/370, you switch the terminal device on and VM/370 responds with the message:

```
vm/370 online
```

to let you know that VM/370 is running and that you can use it. If you do not receive the "vm/370 online" message, see the VM/370: Terminal User's Guide for help. You can now press the Attention key (or equivalent) on your terminal and issue the LOGON command to identify yourself to the system:

```
logon smith
```

where SMITH is used as your userid. The LOGON command is entered by pressing the Return (or Enter) key. If VM/370 accepts your userid, it responds by asking you for your password:

```
ENTER PASSWORD:
```

You then enter your password, which is, in most installations, hidden by the system.

The series of lines you and the computer have exchanged so far looks like this:

vm/370 online	When you switched your terminal on, VM/370 responded to let you know it was ready for your input.
(Attention)	You press the Attention key.
logon smith	You entered the LOGON command and your userid, and then pressed the Return key.
ENTER PASSWORD:	VM/370 asked you for your password in response.
XXXXXXX	You entered your password and the system hides it.

The logging on process is now complete and you can load your virtual machine with an operating system, such as the Conversational Monitor System (CMS).

LOADING CMS IN THE VIRTUAL MACHINE: THE IPL COMMAND

The Conversational Monitor System is an interactive operating system designed for VM/370 virtual machines. CMS provides commands for manipulating virtual disks (virtual disks correspond to real disks on real machines) and files (units of data records such as programs or input data for programs). You load CMS in your virtual machine using the CP IPL command:

```
ipl cms
```

where "cms" is assumed to be the saved system name for your installation's CMS. VM/370 responds by displaying a message such as:

```
CMS VERSION v.l - mm/dd/yy  12:02
```

to indicate that the IPL command executed successfully and that CMS is loaded.

Formatting Your Virtual Disk Storage Space

If this is the first time the virtual disk is used you should make sure that it is formatted. Be careful to use this information only if you know your disk must be formatted. Ask your system programmer whether or not you should format your virtual disk. FORMAT takes the form:

```
format 191 a
```

CMS then prompts you with the following message:

```
DMSFOR603R FORMAT WILL ERASE ALL FILES ON DISK 'A(191)'. DO YOU
           WISH TO CONTINUE? (YES|NO):
```

You answer:

```
yes
```

CMS then asks you for the disk label of the disk to be formatted, which you can get from the system programmer:

```
DMSFOR605R ENTER DISK LABEL:
```

You answer by entering a disk label:

```
MYDISK
```

CMS then erases all the files on that disk, if any existed, and formats the disk for your use. When you enter the label, CMS responds by telling you:

```
FORMATTING DISK 'A'

'3' CYLINDERS FORMATTED ON 'A(191)'.

R; T=0.15/1.60  11:26:03
```

You have to issue this command the first time you use the virtual disk.

The CMS Ready Message

The last line of the response from CMS in the preceding example was a Ready message. All CMS commands generate a Ready message when they complete processing. The Ready message is a response from the system to let you know you can enter data into the system. The form of the Ready message can vary, since it can be changed using the SET command. The preceding example shows the long form of the Ready message. If your Ready message is set to a short form, VM/370 lets you know when you can enter another command by sending you a message that looks like:

```
R;
```

Until now, the commands discussed have had to do with the process of getting you situated on your virtual machine; you have learned to use the terminal, contacted VM/370, logged on, loaded an operating system, and formatted a disk to use. Now you can go on to create a program, assemble, load, and execute it.

The following information is a description of the four steps in writing a program using CMS: (1) creating a program file, (2) assembling the source file, (3) correcting the file if there are errors in it, and (4) loading and executing the program. Before you can begin, however, you should know a little about the CMS Editor.

HOW TO USE THE CMS EDITOR

The CMS Editor is the programming tool you can use to create and modify your program. The information you are about to read tells you how to use the basic EDIT subcommands you need to create your program.

Invoking the Editor: The EDIT Command

The Editor is a CMS component designed to make it convenient to create and modify files using the CMS operating system. The Editor is invoked by the EDIT command, which, for your program, takes the form:

```
edit sample assemble
```

where SAMPLE is the name of the file you are creating, and ASSEMBLE is the filetype of the file. The filetype tells the Editor that the file you are creating is a file written in the assembler language. The Editor sets the correct logical tab settings you should use when entering the Assembler language statements.

When you enter the preceding EDIT command line, CMS responds with the message

```
NEW FILE:
EDIT:
```

You are now in the EDIT mode.

The EDIT Command Modes: EDIT and INPUT

The Editor can be used in two modes, EDIT mode and INPUT mode. The INPUT mode is used solely for entering new lines of data into your file. The EDIT mode is used to correct mistakes you made when you initially created the file or to modify the file. The EDIT command automatically places your virtual machine in the EDIT mode; you enter the EDIT subcommand INPUT to place your virtual machine in the INPUT mode. To get out of INPUT mode, press the Return key (or equivalent) on a null line, that is, press the Return key without entering any data.

Usually, when you create a file, you first create the source records and then check them for errors. The Editor provides subcommands to make the creation and correction of files convenient. The two sections that follow describe (1) the subcommand you use when creating the file and (2) other EDIT subcommands that are useful to you in creating and correcting the program file. All of the facilities of the EDIT command are described in the VM/370: EDIT Guide.

The Current Line Pointer

When you use the Editor, you change one line at a time. The Editor uses a "current line pointer" to point to, or show you, the current line in your file. If you enter the TYPE subcommand, the Editor displays the line it is currently pointing to without changing the pointer.

CREATING YOUR SOURCE RECORDS: THE INPUT SUBCOMMAND

You can use the INPUT subcommand in two ways: (1) to enter the INPUT mode or (2) to insert a single line of data into your file. To enter the INPUT mode, you enter the INPUT subcommand and press the Return key. To insert a single line of data, you enter the INPUT subcommand followed by the line of data, then press the Return key.

When you are in the INPUT mode, the Editor accepts anything you enter on the command line, regardless of its internal system definition. Only the logical editing symbols can be used in the INPUT mode to change a line. All you can do in the INPUT mode is add new lines of data to your file.

Usually the Editor is used to create new files and modify existing files, so it has features that make this convenient, such as line number prompting for some language processors and tab settings for all supported language processors. For the assembler language program you are creating, you can use tab settings provided by the Editor to space to the correct position for assembler language instruction fields.

OTHER EDIT SUBCOMMANDS YOU NEED

The EDIT subcommands described here are almost self-descriptive. They let you see what your file contains, alter its contents, and store it on disk. Remember that these are only a few of the Editor's facilities and that you can find the rest described in the VM/370: EDIT Guide.

To See the First or Last Record of Your File: Use the TOP and BOTTOM Subcommands

To see the first record of your file, instruct the Editor to type the top record:

```
top
```

The top record of the file is always a null line, placed there by CMS so that you can always insert records at the beginning of your file. So, to see the first actual line of data, use the subcommand DOWN 1, which points the Editor to the line below the null line.

To see the last record in your file, instruct the Editor to locate the bottom record of the file:

```
bottom
```

To See Records Above or Below the Current Line: Use the UP and DOWN Subcommands

To see records above the current line, instruct the Editor to move up in your file:

```
up number
```

where number is the number of lines above the active line. For example, UP 7 moves the current line pointer seven lines above the current line.

To see records below the current line, instruct the Editor to move down in your file:

```
down number
```

where number is the number of lines below the current line.

To Delete a Line from Your File: Use the DELETE Subcommand

To delete a line from your file, move up or down in the file so that the line you want to delete is the current line. When the line you want deleted is displayed as the current line, instruct the Editor to delete it:

```
delete number
```

where number specifies the number of lines deleted beginning with the current line. If you enter DELETE 9, the Editor deletes the current line and the next eight lines following it. The new current line is the one following the last deleted line.

To Display Records in Your File: Use the TYPE Subcommand

To display records in your file, enter the TYPE subcommand:

```
type number
```

where number is the number of lines you want displayed, beginning with the current line. The last line displayed then becomes the current line. If you are using a 3270 display terminal in display mode use the SCROLL, FORWARD, and BACKWARD subcommands to display records.

To Change the Contents of a Record: Use the CHANGE Subcommand

To change the contents of a record, issue the CHANGE subcommand:

```
change /string1/string2/
```

where string1 is the string of characters you want to replace and string2 is the string of characters you want to replace them with.

To Store the File and Return to the CMS Environment: Use the FILE Subcommand

To store the commands and the data you have created in your file, instruct the Editor to file your data for you, using the FILE subcommand. Before you issue FILE, however, you must return your virtual machine to the EDIT mode. So press the Return key to enter EDIT mode, and then issue FILE:

```
file
```

All of the records you created are now filed on your virtual disk. The FILE subcommand takes your virtual machine back to the CMS environment. CMS responds to the FILE subcommand with a Ready message, to let you know your virtual machine is back in the CMS environment.

The subcommands just discussed should be adequate for writing and editing a simple program. As you follow the program example you will see how these EDIT subcommands are used.

A SAMPLE PROGRAM

The program you are about to create is a sample program designed to familiarize you with the interactive facilities of VM/370. The program, when it executes, calls for you to enter a line of data in response to a direction from the program. It asks you to enter another item of information. Then, it tells you what you told it.

Enter the program just as you see it. What you enter is shown in lowercase; the Editor responses are shown in uppercase. The first field the Editor allows you to enter is the label field, which begins with column 1. When you press the tab key, the type ball moves to the next tab setting on the terminal, and this lets the Editor know that the next field you enter goes in the operation field, which begins in column 10. When you press the tab key again, the Editor spaces to the operand field, beginning in column 16.

There are several ways you can get a continuation mark in column 72; see the VM/370: EDIT Guide for this information. In this example, the $MARK edit macro is used. You must add the $MARK edit macro to your CMS system; it is not distributed with CMS. The VM/370: EDIT Guide tells you how to add it to your CMS system.

Note that the example shows one space after the third LINEDIT macro instead of pressing the tab key. The third LINEDIT macro line is so long that is does not fit if you tab to column 31. When you enter the third LINEDIT macro you must press the Return key twice after the last comma: once to enter the line and one to return to EDIT mode. You know you are in EDIT mode when EDIT: is displayed. Then you must enter the macro '$MARK'. This places an * continuation mark in column 72. To return to INPUT mode, enter 'input'.

```
edit myfile assemble
NEW FILE:
EDIT:
tabs 1 10 16 31 36 41 46 69 72 80
input
INPUT:
When INPUT: is displayed, you continue entering your program.
sample   csect
         using sample,r12     set up addressability
         lr    r12,r15        load base register
         st    r14,savret     save return address
         linedit              text='please enter your name'
         rdterm               name
         linedit              text='please enter your age'
         rdterm               age
         linedit text='hi, ..........., you just told me you are ....',

EDIT:
$mark
input
INPUT:
               sub=(chara,name,chara,age),rent=no
         l     r14,savret     get return address
         br    r14            return to caller
         eject
name     dc    cl130'    '    name field
age      dc    cl130'    '    age field
savret   dc    f'0'           save return address
         regequ
         end
```

When you finish entering the program, you can display it, and then you should file it. If you display the program, it should be:

```
SAMPLE   CSECT
         USING SAMPLE,R12     SET UP ADDRESSABILITY
         LR    R12,R15        LOAD BASE REGISTER
         ST    R14,SAVRET     SAVE RETURN ADDRESS
         LINEDIT              TEXT='PLEASE ENTER YOUR NAME'          Col
         RDTERM               NAME                                    72
         LINEDIT              TEXT='PLEASE ENTER YOUR AGE.'           |
         RDTERM               AGE                                     V
         LINEDIT TEXT='HI, ..........., YOU JUST TOLD ME YOU ARE ....', *
               SUB=(CHARA,NAME,CHARA,AGE),RENT=NO
         L     R14,SAVRET     GET RETURN ADDRESS
         BR    R14            RETURN TO CALLER
         EJECT
NAME     DC    CL130'    '    NAME FIELD
AGE      DC    CL130'    '    AGE FIELD
SAVRET   DC    F'0'           SAVE RETURN ADDRESS
         REGEQU
         END
```

Creating and Executing Your Program

This section describes the CMS commands you use to create, load, and run your sample program: EDIT, GLOBAL, ASSEMBLE, LOAD, and START. These commands represent the most basic method of creating and executing a program using CMS.

INPUT FOR THE PROGRAM

To create the program file, enter the EDIT command to invoke the CMS Editor.

```
edit sample assemble
```

The Editor responds to the command by telling you that this is a new file and that you can begin creating it

```
NEW FILE:
EDIT:
```

You then place your virtual machine in the INPUT mode by entering the INPUT subcommand

```
input
```

When you press the Return key, the Editor responds with the message

```
INPUT:
```

Now you can enter the program statements. When you have entered all of your program statements, press the Return key to return to the EDIT environment. Now issue the FILE subcommand to save the statements you have entered and to return to the CMS command environment.

ASSEMBLING YOUR PROGRAM

To assemble SAMPLE, you issue two commands: GLOBAL and ASSEMBLE. The GLOBAL command accesses libraries needed by the assembler during its execution. The ASSEMBLE command causes your program to be assembled by the assembler. You should enter both GLOBAL and ASSEMBLE just as they are shown here:

```
global maclib cmslib
assemble sample
```

CMS notifies you when the assembly is done by displaying a message at your terminal. You receive one of two kinds of messages: one to tell you there were no errors, the other to tell you there were errors and where they were in the program. In either case, a Ready message is generated to tell you that ASSEMBLE has completed execution.

If there are no errors, the message is

```
ASSEMBLER DONE
NO ERRORS FLAGGED IN THIS ASSEMBLY
R;
```

If you have errors, you receive a message like

```
ASSEMBLE DONE
SAM00331          B        ERRT
IFO024   NEAR OPERAND     COLUMN1--UNDEFINED SYMBOL
1 STATEMENT WAS FLAGGED IN THIS ASSEMBLY
8 WAS HIGHEST SEVERITY CODE
R(008);
```

If you receive the error message, you have to go back and correct your program; the assembler language error message you receive tells you where and how you made the mistake. Use the EDIT command to correct the file:

```
edit sample assemble
```

This statement allows you to correct your source file. The Editor places your virtual machine in the EDIT mode and places the file in storage so you can edit it. You can now begin making corrections using the EDIT subcommands described in the first part of this section.

When you have corrected the program, file it again and reassemble it (assemble sample). This time you need not enter the GLOBAL command, since the first GLOBAL command is in effect for the duration of your terminal session.

If the program assembles with no errors, you can load it in virtual storage and execute it.

LOADING AND EXECUTING THE PROGRAM

The CMS LOAD and START commands load your program in virtual storage and begin its execution. The LOAD command processes the relocatable object code produced by the ASSEMBLE command. The LOAD command loads the assembler TEXT file into storage and establishes the proper linkage. Enter the LOAD command as it is shown here:

```
load sample
```

After you press the Return key, CMS responds with a Ready message (R;) when your program is loaded successfully.

Now enter the START command to begin execution of the program:

```
start sample
```

The virtual machine, with your program now in control, prompts you with the messages supplied by your program:

```
PLEASE ENTER YOUR NAME:
PLEASE ENTER YOUR AGE:
```

After you answer its questions, your program writes you a message and returns a Ready message.

LOGGING OFF

When you want to end the terminal session, use the LOGOFF command,

```
logoff
```

Section 6: Virtual Machine Operation

Virtual System Console

The major differences between operating on the real machine and operating under CP are:

- You can enter CP commands by entering CP mode, or by issuing the #CP command while a virtual machine's console is able to read.
- The Request key of a real operator's console is simulated by signalling attention once from the terminal.

To reply to messages from a virtual machine operating system, you:

- Must enter VM mode (if you on a 2741 communication terminal).
- Enter the information requested.
- Press the carriage return (Return) key on the 2741 Communication Terminal (or equivalent on other terminals).

If communication with CP is required before responding to specific error messages issued by some virtual machine operating systems, enter CP mode and perform the necessary console functions. When control returns to the virtual machine environment, the virtual machine read is canceled by a unit exception and the virtual machine operating system responds by reissuing the read. The required virtual machine response may then be entered.

The Cancel key of the system console keyboard is simulated by the Attention key on the 2741. To enable the CPU operator to make input line and character corrections to operating system responses, the VM/370 logical editing symbols may be used.

Attaching Devices

When a virtual machine operating system requires a specified device, as in a MOUNT request, you should do one of the following:

- If the device is already defined as part of the virtual machine configuration, enter CP mode. Then enter the CP command READY cuu (with cuu replaced by the virtual I/O device address), followed by the CP BEGIN command. This is the virtual machine equivalent of physically making a device ready, if it is not already in a ready state.
- If the device is not a standard device already defined as part of the virtual machine configuration, enter CP mode, and issue the CP DEFINE command to add the device to the virtual machine configuration. If a DASD device is already defined for some other user in the VM/370 directory, you can use the CP LINK command to add the device to your virtual machine configuration. Otherwise, send a message to the

system operator to attach it to your virtual machine. When the real device is ready, enter the BEGIN command.

Note: If the virtual device is an attached real device at the time you need it mounted, the real device can be taken out of and put back into ready status to cause an interrupt.

TAPE DEVICES

Tape drives, because they cannot be shared, are usually not defined in the VM/370 directory as part of a virtual machine configuration. When you require a tape drive for a job, a system operator with privilege class B must attach an available real device to your virtual machine with the virtual device address you require. When the job involving that tape drive is complete, you can issue a DETACH command, naming the virtual device address of the tape to release the tape. In this way, real tape drives are allocated to a virtual device only for the duration of the jobs that need them.

Loading an Operating System into a Virtual Machine

Figure 10 shows how you could load an OS/MFT system on a virtual machine, and the subsequent mounting of a virtual device at address 232. (For clarity, the information you enter is shown in lower case, and messages displayed by CP and OS are shown in uppercase).

The virtual machine that is to run OS must have a configuration compatible with that for which the operating system's nucleus was generated. An example of an appropriate VM/370 directory entry is:

```
USER BATCH PASSWORD 512K
 ACCOUNT NUMBER BIN10
  OPTION REALTIMER ISAM
  CONSOLE 01F 3215
  SPOOL C 2540 READER A
  SPOOL D 2540 PUNCH A
  SPOOL E 1403 A
  DEDICATE 230 BATCH1
  DEDICATE 231 BATCH2
```

```
logon os/mft
ENTER PASSWORD:
(Printing of the password is inhibited)

LOGON AT 13:24:30 EST THURSDAY  mm/dd/yy
ipl 230                                  IPL the OS system residence device.
.                                        You must respond to all OS IPL commands.
.
.
IEE007A READY                            OS ready message.
set date = 72.355,Q =  (231)

start rdr, 00c
start wtr, 00e
start
IEF233A M 232, VOLABC,JOB,STEP           OS requests operator to mount a pack labeled
                                         "VOLABC" on 232.

!                                        You press the attention key once to present
                                         an interrupt to OS.

#cp link to usera 330 as 232 w pass      Because the LINK command was prefaced by #CP,
                                         the OS virtual machine acts as though you
DASD 232 LINKED TO R/W                   had pressed the CANCEL key, and issues
                                         another READ to the system console. The LINK
                                         command is passed directly to CP.
                                         You request that a disk belonging to usera,
                                         with a virtual address of 330, be linked
                                         to your own virtual machine.  You issue the
                                         CP LINK command to access that other user's
                                         disk, with virtual address 232 in his virtual
                                         machine. Alternately, you could issue the
                                         "vary 232,online" OS command.

#cp ready 232                            Then enter READY to simulate a ready
                                         interrupt to OS and end the read by pressing
                                         the carriage return or Enter key.
```

Figure 10. Loading an OS/MFT Virtual Machine

Spooling Virtual Console I/O

CP lets you spool your virtual machine's console input/output to disk, instead of, or in addition to, having it displayed at your terminal. The data spooled includes messages from or to the virtual machine operating system, and from or to CP. This facility, invoked by the SPOOL CONSOLE command, is particularly useful when the virtual machine is executing with the terminal disconnected, or when the virtual machine console is a display device, since the virtual console output, which would otherwise be lost, is saved on disk. The saved data is later printed on the real printer.

If the real printer cannot print lower-case characters, the operator can use the FOLD option when he loads the buffer for the print train image. This causes the printer to print, in upper case, data that you enter in lower case.

You can invoke or terminate virtual console I/O spooling at any time and as often as you like. If the console file is not closed when you log off (or are forced off) the VM/370 system, CP closes the console spool file and schedules it for printing.

Reading Cards in a Virtual Machine

When an operating system such as DOS, DOS/VS, or OS/PCP, MFT, MVT, VS1, or VS2 is running batch production jobs in a virtual machine, it may occasionally be necessary to dedicate a real card reader to that virtual machine. In that case, jobs are entered through the card reader in exactly the same way they are entered on a standalone system, and no double spooling occurs.

If, however, there is no extra reader available to dedicate in such a manner, or if the virtual machine is being run from a terminal not located near a real card reader, other methods of handling card input are necessary.

Card images can be placed into the (spooled) virtual card reader of your virtual machine in four ways:

- A card deck can be placed in the real card reader and read by VM/370. The deck must be preceded by a special VM/370 identification (ID) card specifying your virtual machine userid. VM/370 reads the cards and transfers the card images to your virtual machine's virtual card reader as a spool file. When your virtual machine issues a read to the card reader, VM/370 presents the cards, one at a time, to the virtual machine. The format of the ID card is described in the CMS READCARD command description.

- A virtual machine's printer can transfer printer listing images and the punch can transfer punch images to its own (or to some other user's) virtual card reader. CP can handle up to 150-character records in virtual card readers. The CP SPOOL transfers these unit record files. For example:

  ```
  spool punch to usera
  ```

  causes all cards punched to be transferred (after the punch file is closed) as an input file to usera's spooled card reader.

  CMS cannot print files that have records greater than 160 characters long. However, CMS can display records greater than 160 characters long.

- Via the CP TRANSFER command, you can transfer your virtual reader file(s) to another user's virtual reader, or retrieve previously transferred files back to your own virtual reader.

- A card deck can be placed in the card reader of any of the remote stations supported by the Remote Spooling Communications Subsystem (RSCS). The same VM/370 ID control card, as mentioned previously, must precede the deck. RSCS receives the card file and spools it to the specified VM/370 user's virtual card reader.

Normally you should find it helpful to set the following spooling options for your virtual card reader:

```
spool 00c cont noeof
```

where:

cont specifies that reading be "continuous" (that is, not to indicate an EOF condition to the virtual machine after each input file, but to continue reading all the card images until the reader

files spooled to the virtual machine are exhausted). If this option is not in effect, a unit exception is reflected to your virtual machine at the end of each spooling file. This option eliminates the need to repeatedly enter the CP READY and CLOSE commands between the reading of each spooled file.

noeof indicates that the virtual reader's end-of-file button is assumed not to have been pushed. When all reader spool files are exhausted, an intervention-required status pending is reflected to the virtual machine. When additional reader files are transfered to the virtual card reader, a device end interrupt is reflected to the virtual machine, and card reading normally resumes automatically (because a device end interrupt "wakes up" the virtual machine).

The operator of a virtual machine operating system such as OS or DOS may choose to prepare his own job streams at his terminal using the CMS Editor facilities, and then IPL another operating system to process the job streams. This procedure is discussed under "Using Multiple Consecutive Operating Systems."

Printing and Punching in a Virtual Machine

VM/370 provides unit record spooling facilities. When operating systems that also provide unit record spooling facilities are running in a virtual machine, double spooling of printed and punched output may occur. If you wish, you can eliminate one of the spooling facilities.

To eliminate CP spooling, a real printer or punch can be temporarily attached to a virtual machine, or dedicated to a virtual machine in the VM/370 directory via the DEDICATE statement. In either case, CP performs no spooling for that virtual device. Interrupt processing can be initiated by the virtual machine whenever it receives control of the CPU from CP.

CP spooling has priority over any virtual machine execution. The real printer and punch are better utilized by CP than by a virtual machine. CP spooling provides more efficient shared use of a limited number of readers, printers, or punches by many virtual machines.

OS spooling can be eliminated by using OS JCL in the form UNIT=00E or UNIT=00D instead of SYSOUT=A or SYSOUT=B, where 00E and 00D are virtual device addresses.

DOS POWER spooling can be eliminated by using the Job Entry Control Language (JECL) option of POWER and by indicating no spooling on the JECL control statements for dedicated spooling devices.

When printer or punch spooling is performed by CP, the CP CLOSE command specifying the virtual printer or punch address should be issued periodically. This releases previously finished and stacked output files for CP spooling and results in better utilization of the DASD spooling areas and of the unit record devices.

CMS has no spooling facilities of its own, and does not cause double spooling when running in a virtual machine. It does not support an attached or dedicated printer, punch, or reader, and must use the facilities of CP. CMS closes the virtual printer or punch after the completion of each command that uses them.

Disconnecting the Terminal

Once you load a virtual machine operating system and start jobs you may want to use your terminal for some other purpose while the batch jobs are running. The DISCONN command allows you to disconnect your terminal from the VM/370 system, but allows the virtual machine to continue operation. You can reconnect your terminal by issuing the LOGON command.

Unless the CP command, SPOOL CONSOLE START, is issued to spool the virtual console and CP output, all "writes" or output messages to the virtual console are ignored.

A HOLD option is provided with the DISCONN command to prevent a communication line in a switched line configuration from being disabled.

Disconnecting the terminal frees it for other uses, but is useful only for operating systems that can run in an unattended mode. If a virtual machine that is running disconnected issues a "read" to the virtual console or enters a disabled wait state, virtual machine execution is halted and a 15-minute time-out begins. If you do not re-establish your connection to the VM/370 system (by logging on again) within 15 minutes, you are logged off the system.

A virtual machine may also be placed in disconnect status automatically. If a teleprocessing line error occurs, or you turn off your terminal without logging off, the virtual machine is disconnected and execution stops. (If you turn off a 3270 without logging off, the virtual machine is disconnected if someone sends a message to the terminal.) A 15-minute time-out begins. If you do not log on again within 15 minutes, your virtual machine is logged off the system.

When a disconnected machine is logged off, VM/370 closes all open spool files and schedules them for printing or punching.

You reconnect to a disconnected virtual machine via the normal logon procedure. (If there was a read active on the terminal at the time of disconnect, you may have to issue a RESET command for the terminal.) Your running virtual machine is placed in the CP command mode. To resume execution of the virtual machine operating system, issue the BEGIN command.

For example, (a) shows the disconnect procedure, and (b) shows the subsequent reconnect, as long as PROGA is still running when USERA reconnects.

```
        (a)
logon usera
ENTER PASSWORD:
(displaying of the password is inhibited)
LOGON AT 05:30:30 EST FRIDAY mm/dd/yy
ipl 190
CMS VERSION 2 LEVEL 0
run proga
•••
CP
disconn
DISCONNECT AT 05:35:30 EST FRIDAY mm/dd/yy
```

```
        (b)
    logon usera
    ENTER PASSWORD:
    (displaying of the password is inhibited)
    RECONNECT AT 07:42:45 EST FRIDAY mm/dd/yy
    begin
```

Note: A PROFILE EXEC procedure (if one exists) is not executed automatically when you log on after disconnecting. Also, if you set a system option (such as specifying TERMINAL APL ON) before you disconnect, that option may be returned to its default when you log on again.

Using Multiple Consecutive Operating Systems

You may require the facilities of more than one operating system during a single terminal session.

When you run an operating system such as OS, DOS, OS/VS1, OS/VS2, or DOS/VS from a terminal, you can use the CMS Editor facilities to create and modify job streams.

If you are an applications programmer who normally uses CMS to interactively create, modify, and test your programs you may require facilities not supported or available in CMS for compilation (for example, programs using DOS macros) or for execution (for example, any DOS object program, or any OS object program that utilizes certain OS supervisory functions and access methods not simulated under CMS).

The following technique uses multiple operating systems consecutively. Job control cards, compiler or assembler source programs, and test data streams are created and modified at the terminal under control of the CMS Editor. Then the job stream is executed, only after control has been passed to an appropriate operating system with the necessary facilities.

In this way, you can use the terminal-oriented facilities of CMS to create and update source programs and JCL. When you are ready to compile or test, you can give control of your virtual machine to DOS or OS. After execution is finished, you can transfer the printer and punch output back to CMS for selective scanning and displaying at the terminal.

This approach assumes you have created source program files and data files under CMS. To execute under another operating system (in this example, OS) you must also create JCL records that specify the compilation, link edit, or execution, as appropriate. These records are created under CMS and named with a distinctive filename and filetype (for example, PLICOMP JCL). Job control records, source program files, and data files can then be merged together in the virtual card reader to form a single OS job stream. CP and CMS commands (shown in Figure 11) create and transfer this job stream.

TRANSFERRING OUTPUT

The CP SPOOL command transfers card images from the virtual card punch of one virtual machine to the virtual card reader of that same or some other virtual machine. During this time, no real cards are punched or read; CP manages the transfer of CMS card-image data files through disk spooling operations only.

Figure 11 shows how files are transferred between virtual machines. The virtual machine is in CMS mode at the start of the example. The command "SPOOL 00c cont eof" specifies that reading be continuous until all files spooled to the virtual machine are exhausted and the virtual end-of-file button on the reader is pushed. NOHEADER specifies that no special control cards are to be inserted in front of each punched file. Virtual device 230 is an OS system volume. Virtual device 231 contains the OS job queue, SYS1.SYSJOBQE. All standard OS responses (for example, R;T=0.04/0.12 09:36:08) are omitted from the example; however, the OS READY message is included to more fully illustrate the IPL sequence. Also, assuming you are using a 2741, you must press the Attention key before entering each OS command. The attention interrupts are not shown in Figure 11.

```
CMS
cp close 00c
cp close 00d
cp spool 00d to *
punch jobcard jc1 (noheader)
punch plicomp jc1  (noheader)
punch plimain pl1  (noheader)
punch asmcomp jc1  (noheader)
punch asmsub assemble  (noheader)
punch linkgo jc1  (noheader)
punch godata dat (noheader)
punch slshstar jc1 (noheader)
cp spool 00c cont eof
cp ipl 230

Note:  The following are issued once under OS control:

IEE007A READY
set date=xx.355,Q=(231)
start rdr,00c
start wtr,00e
start
```

Figure 11. OS Job Stream Transfer

To transfer files between virtual machines, you must have access to both operating systems being used. Access to both systems can be provided either in your virtual machine's VM/370 directory entry, or dynamically before loading the new system.

Figure 12 illustrates a virtual machine configuration and the corresponding VM/370 directory control statements. Virtual device addresses 190 and 191 contain the CMS system and user disk area. Virtual device addresses 230 and 231 contain the OS system and user disk area. The two systems use a common card reader, card punch, printer, and console.

```
USER OS2 PASSWORD
 ACCOUNT NUMBER BIN16
  CONSOLE 01F 3215
  SPOOL  C 2540 READER
  SPOOL  D 2540 PUNCH
  SPOOL  E 1403
  LINK JFK 230 230 R
  LINK CMSSYS 190 190 RR
  MDISK 231 2314 120 82 W
  MDISK 191 2314 101 10 UDISK1 WR RPASS WPASS
```

Figure 12. Directory Entry for Alternating Operating Systems

CONFIGURATIONS

You can alternate operating systems more simply if:

- The devices used by both systems are supported at the same device address, and
- The common addresses are not used to support different devices

If these two conditions do not exist, you must modify the virtual machine configuration before each IPL of a new system.

If the two systems require online typewriter keyboards at different addresses, the CP DEFINE command can be used to change the address of the virtual system console. For example, if the OS system specified in Figure 12 required an online typewriter keyboard at address 01F, issue the command

```
cp define 009 as 01F
```

before loading 230. When the OS job stream is completed, issue the command

```
cp define 01F as 009
```

before loading CMS. Virtual storage size can be changed and virtual card readers, printers and punches can be added with this procedure.

If the systems expect different device types at the same address (for example, in Figure 12 CMS expects a 2314 at address 191, but the OS system might be generated to support a 3330 at that address), the common address must be assigned to the appropriate device each time a new system is loaded. If CMS is running with a disk at address 191, you should issue the following command before loading OS:

```
cp detach 191
```

An appropriate device can then be added to the virtual machine at address 191 either before loading, or in response to a mount request from the OS system.

Note: For direct access storage devices, the above procedure is necessary even if both systems support the same device type at the same address. The disk format used by CMS is unique, and is not compatible with that of other operating systems. Files can be shared between CMS and OS or DOS only through the spooling facilities of CP.

A string of arguments can be passed to your program from the terminal when execution is begun with the CMS START command. If arguments are specified, the storage address of a parameter list is placed in general register 1. The parameter list is a string of double words, one argument per double word. The first argument is the entry operand. Other arguments are accessed with displacements of 8, 16, 24, and so on, from the address contained in register 1 when execution begins. For example:

```
load proga
start epg  071374
```

causes execution to begin at a control section or entry point named EPG. When execution begins, general register 1 points to a string of two double words, the first containing EPG, the second containing 071374.

If a program is to be executed frequently, you can create a non-relocatable copy of it on one of your disks. Subsequent invocation of this program causes the absolute module to be read from disk storage, ready to begin execution.

Part 2: Reference Information

This part of the publication contains reference information. Section 7 contains all the formats and rules for using CMS commands. Section 8 contains the formats and rules for using CP commands.

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 VM/370 commands can be entered with any combination of upper and lowercase letters.) The example below shows the format specification for the FILEDEF command.

  ```
  FIledef
  ```

  This representation 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 capital letters) must be entered.

  Abbreviations are shorter forms of command names, operands, and options. Abbreviations for command names are shown below the full name in the format box. Abbreviations for operands and options are shown in the description of the individual operands and options that follows the format box. For example, the operand READER has both a minimum truncation and an abbreviation. In the format box it is shown as:

  ```
  Reader
  ```

  indicating that the minimum truncation is R. In the discussion of the READER operand that follows, it is shown as:

  ```
  READER
  RDR
  ```

  indicating that the abbreviation is RDR. Thus, the acceptable specifications for the READER operand are: R, RE, REA, READ, READE, READER, and RDR.

  In some cases what appears to be a minimum truncation is really the only valid abbreviation. For example, the abbreviation for MEMBER 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	:

- 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

  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

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

- 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 Summary

This section contains descriptions of the commands acceptable in the CMS environment. Figure 13 contains an alphabetical list of the CMS commands and the functions performed by each. Unless otherwise noted, CMS commands are described in this manual.

Command	Usage
ACCESS	Define direct access space for a CMS virtual machine, create extensions and relate the disk space to a logical directory.
ASM3705[1]	Assemble 3704/3705 source code.
ASSEMBLE	Assemble Assembler Language source code.
CMSBATCH	Invoke the CMS Batch Facility.
COBOL[2]	Compile ANS Version 4 COBOL source code.
COMPARE	Compare all or part of records in two existing disk files.
CONVERT[2]	Convert free form FORTRAN statements to fixed form.
COPYFILE	Copy files according to specifications.
CP	Enter CP commands from CMS environment.
CPEREP[3]	Dump error information which was recorded by VM/370 error recording routines.
DDR[1,3]	Perform backup, restore, and copy operations for virtual disks.
DEBUG[4]	Enter DEBUG subenvironment, DEBUG mode.
DIRECT[1,3]	Set up VM/370 directory entries.
DISK	Perform disk-to-card and card-to-disk operations for CMS files.

[1]This command is described in the VM/370: Planning and System Generation Guide.
[2]This command invokes an IBM Program Product, available from IBM for a license fee.
[3]This command is described in the VM/370: Operator's Guide.
[4]This command is described briefly in this manual and in detail in the VM/370: System Programmer's Guide.

Figure 13. CMS Command Summary (Part 1 of 4)

Command	Usage
EDIT[5]	Enter EDIT subenvironment, EDIT mode.
ERASE	Delete files from user disks.
EXEC[6]	Process special procedures made up of frequently used sequences of commands.
FILEDEF	Provide simulation of OS job control language data definition (DD) statements.
FORMAT	Prepare disks in CMS 800-byte block format.
FORTGI[2]	Compile FORTRAN source code using the G1 compiler.
FORTHX[2]	Compile FORTRAN source code using the H-extended compiler.
GEN3705[1]	Generate an EXEC file that assembles and link edits the 3704/3705 control program.
GENDIRT	Create auxiliary module directories.
GENMOD	Generate absolute non-relocatable files (MODULE files).
GLOBAL	Define specific CMS libraries to be searched for macros and subroutines.
GOFORT[2]	Compile FORTRAN source code and execute the program just compiled using Code and Go compiler.
INCLUDE	Bring additional TEXT files into storage and establish linkage.
LISTDS	List information about data sets on an OS disk.
LISTFILE	List information about user CMS files.
LKED[1]	Link edit the 3704/3705 control program.
LOAD	Bring TEXT files into storage and establish linkages.
LOADMOD	Bring a single MODULE file into storage.
MACLIB	Create and modify CMS macro libraries.

[1]This command is described in the VM/370: Planning and System Generation Guide.
[2]This command invokes an IBM Program Product, available from IBM for a license fee.
[5]This command is described briefly in this manual and in detail in the VM/370: EDIT Guide.
[6]This command is described briefly in this manual and in detail in the VM/370: EXEC User's Guide.

Figure 13. CMS Command Summary (Part 2 of 4)

Command	Usage
MODMAP	Display load map of a MODULE file.
MOVEFILE	Move data from one device to another device of the same or different type.
NCPDUMP[1,3]	Process CP spool reader files created by 3704/3705 dumping operations.
PLIC[2]	Compile the PL/I source code using the PL/I Checkout Compiler.
PLICR[2]	Execute the PL/I object code generated by the PL/I Checkout Compiler.
PLIOPT[2]	Compile the PL/I source code using the PL/I Optimizing Compiler.
PRINT	Spool a specified CMS file to the virtual printer.
PUNCH	Spool a specified 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.
RUN	Initiate series of functions to be performed on a file.
SAVENCP[1]	Read 3704/3705 control program load into virtual storage and save an image on a CP-owned disk.
SCRIPT[7]	Format and print documents according to embedded SCRIPT control words in the document file.

[1]This command is described in the VM/370: Planning and System
[2]This command invokes an IBM Program Product, available from IBM for a license fee.
[3]This command is described in the VM/370: Operator's Guide.
[7]This command invokes a text processor that is an IBM Installed User Program, available from IBM for a license fee.

Figure 13. CMS Command Summary (Part 3 of 4)

Command	Usage
SET	Establish, set, or reset CMS virtual machine characteristics.
SORT	Arrange a specified file in ascending order according to specified fields in the data record.
START	Begin execution of programs previously loaded.
STATE	Verify the existence of a file.
SVCTRACE	Record information about supervisor calls.
SYNONYM	Invoke a table containing synonyms you have created for CMS commands.
TAPE	Perform tape-to-disk and disk-to-tape operations for CMS files.
TAPPDS	Load OS partitioned data set (PDS) files or card image files from tape to disk.
TESTCOB[2]	Execute the OS COBOL Interactive Debug Program Product.
TESTFORT[2]	Execute the FORTRAN Interactive Debug Program Product.
TXTLIB	Generate and modify text libraries.
TYPE	Display all or part of a file at the terminal.
UPDATE	Make changes in a file as defined by control cards in a control file.
VMFDUMP[3]	Format and print system ABEND dumps.
VSBASIC[2]	Compile and execute VS BASIC programs under CMS.
VSBUTIL[2]	Convert BASIC 1.2 data files to the format required by VS BASIC.
ZAP[1,3]	Modify or dump LOADLIB, TXTLIB, or MODULE files.

[1]This command is described in the VM/370: Planning and System Generation Guide.
[2]This command invokes an IBM Program Product, available from IBM for a license fee.
[3]This command is described in the VM/370: Operator's Guide.

Figure 13. CMS Command Summary (Part 4 of 4)

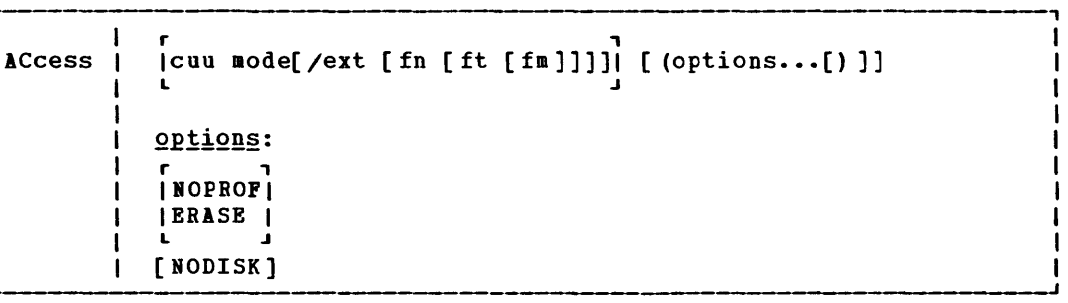
Any of the commands listed in Figure 13 may be entered when you are running CMS in your virtual machine, the terminal is idle, and the virtual machine is receptive for input. If however, CMS is processing a previously entered command and your typewriter terminal keyboard is locked, you must signal your virtual machine via an attention interrupt. The system acknowledges the interrupt 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 most recent command is stacked and is not executed until the command that is currently executing completes.

In addition to the commands listed in Figure 13, 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 executing by pressing the Attention key (or its equivalent) and are executed immediately. The Immediate Commands are:

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

ACCESS

Use the ACCESS command to acquire disk space and to set up file directories in storage to be used during a terminal session. A Master File Directory is the disk-resident file directory containing an entry for each CMS file on a virtual disk. A User File Directory is the directory created in storage when the ACCESS command is executed. It contains an entry for each CMS file that is available to you in your CMS virtual machine. The specifications of the ACCESS command determine the entries in the User File Directory. The format of the ACCESS command is:



where:

cuu — makes the disk at the specified virtual device address available. This field must be specified if other than the default value is desired. The default value is 191, except when the NODISK option is specified. (When NODISK is selected, the only disk accessed is the system disk and any of its extensions.)

Note that 000 is not a valid address.

mode — assigns the one-character filemode letter to the disk being accessed. This field must be specified if cuu is specified (except when the NODISK option is selected). The default value is A.

ext — indicates the mode of the parent disk. The User File Directory of this disk is logically associated with the User File Directory of the read-only extension. There must not be a blank preceding or following the slash (/).

fn — searches the Master File Directory of the disk being accessed and includes the files with the specified filename in the User File Directory for that disk. An asterisk coded in this field means that all filenames are to be included. (See Note 5.)

ft — searches the Master File Directory of the disk being accessed and includes the files with the specified filetype in the User File Directory for that disk. If an asterisk is coded in this field, all filetypes are included. (See Note 5.)

fm — searches the Master File Directory of the disk being accessed and includes the files with the specified filemode in the User File Directory for that disk. An asterisk coded in this field means that all files on the disk specified by the filemode letter and device address are selected, regardless of the filemode number. (See Note 5.)

Note: An asterisk (*), preceded by any number of characters for filename or filetype, allows the specified characters to be used as the leading characters for that identifier. For example, ABC* for fn allows access to all files with filenames beginning with ABC.

Options

NOPROF suppresses execution of a PROFILE EXEC file (see Note 1). This option is valid only if the ACCESS command is the first command entered after an IPL of the CMS operating system. On subsequent ACCESS commands, the NOPROF option is ignored.

ERASE creates a User File Directory with no entries for the disk being accessed; this option is valid only for disks in read/write mode. The Master File Directory on the disk remains unchanged; only the directory in virtual storage is altered (see Note 3).

NODISK lets you gain access to the CMS operating system with no disks accessed except the system disk and its extensions. This option is valid only on the first command you enter after loading CMS. If NODISK is specified, no other operands or options may be used.

Notes

1. The PROFILE EXEC file is a user-generated EXEC procedure often used to perform initialization and to set system parameters required by the virtual machine. For example, whenever programs that use macros are assembled or compiled, the PROFILE EXEC procedure can issue the GLOBAL command to indicate where the system should search for the macros. For OS programs, the PROFILE EXEC procedure can issue FILEDEF statements for the files to be used.

2. If an ACCESS command is not entered as the first command after an IPL command for CMS, the command:

```
access 191 a
```

   is automatically performed and the PROFILE EXEC file, if one exists, is executed. Any disk being accessed must previously have been formatted using the CMS command FORMAT.

3. If an ACCESS command with the ERASE option is entered by mistake, you can regain access to the existing files on the disk either by issuing another ACCESS command without the ERASE option, or by issuing a RELEASE command for the disk. Since the ERASE option does not alter the Master File Directory or disk files, but only specifies that the directory built in virtual storage is to contain no entries, the original disk files can be recovered as long as they were not altered. However, if a CMS command is executed which causes a new file to be written on the disk, the Master File Directory is updated and the files which were on the disk before the execution of the ACCESS command with the ERASE option are no longer available. The space previously used for the erased files is now available for new files.

4. If the initial ACCESS command (either explicit or implied) encounters an error, a message is displayed at the terminal and you can enter another command; no disk has been accessed at this time.

5. The filename, filetype and filemode can only be specified with disks which are accessed as read-only extensions. Files with a mode number of 0 are not be accessed in this case.

6. 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. The S and Y disks are read-only disks.

7. If you are using ACCESS to access a read-only OS or DOS disk, you cannot specify the fn, ft, or fm operands, nor can you specify any options.

8. A read-only OS or DOS disk cannot be accessed unless a CMS A-disk is already accessed, in read/write mode.

9. Only one virtual machine at a time can access a disk in read/write mode.

Examples

Command	Result
ACCESS 192 B	Makes all files on 192 (which is to become the B-disk) available for reading.
AC 192 B/B ABC	Provides read-only access to all files on the B-disk with a filename of ABC.
ACC 192 B/B ABC*	Provides read-only access to all files on the B-disk whose filenames have ABC as the first three characters.
ACCE 192 B/B ABC * B2	Provides read-only access to all files on the B-disk that have a filename of ABC and that are in read-only mode (mode number of 2).
ACCES 191 A/A	Places the A-disk in read-only mode.
ACCESS 194 D/A	Makes the D-disk a read-only extension of the A-disk. The A-disk must have been accessed before this command is issued. The effect of this can be negated by issuing ACCESS 194 D
ACC (NOPROF)	Accesses 191 as the A-disk and does not execute your PROFILE EXEC file if this is the first command executed after you load CMS.
AC 191 A (ERASE)	Makes 191 available as the A disk but builds the file directory in virtual storage with no entries. All space on the disk becomes available for new files, when the execution of a CMS command causes the Master File Directory to be updated.

Responses

DMSACC723I mode (cuu) R/O [-OS]

The specified disk is attached to the CMS virtual machine in read-only mode. When the disk specified is in OS or DOS format, CMS adds the "-OS" to the response.

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]

The disk specified by cuu is the "mode" disk and an ACCESS command was issued to assign it another filemode letter. When the disk specified is in OS or DOS format, CMS adds the -OS to the response.

DMSACC726I 'cuu mode' RELEASED

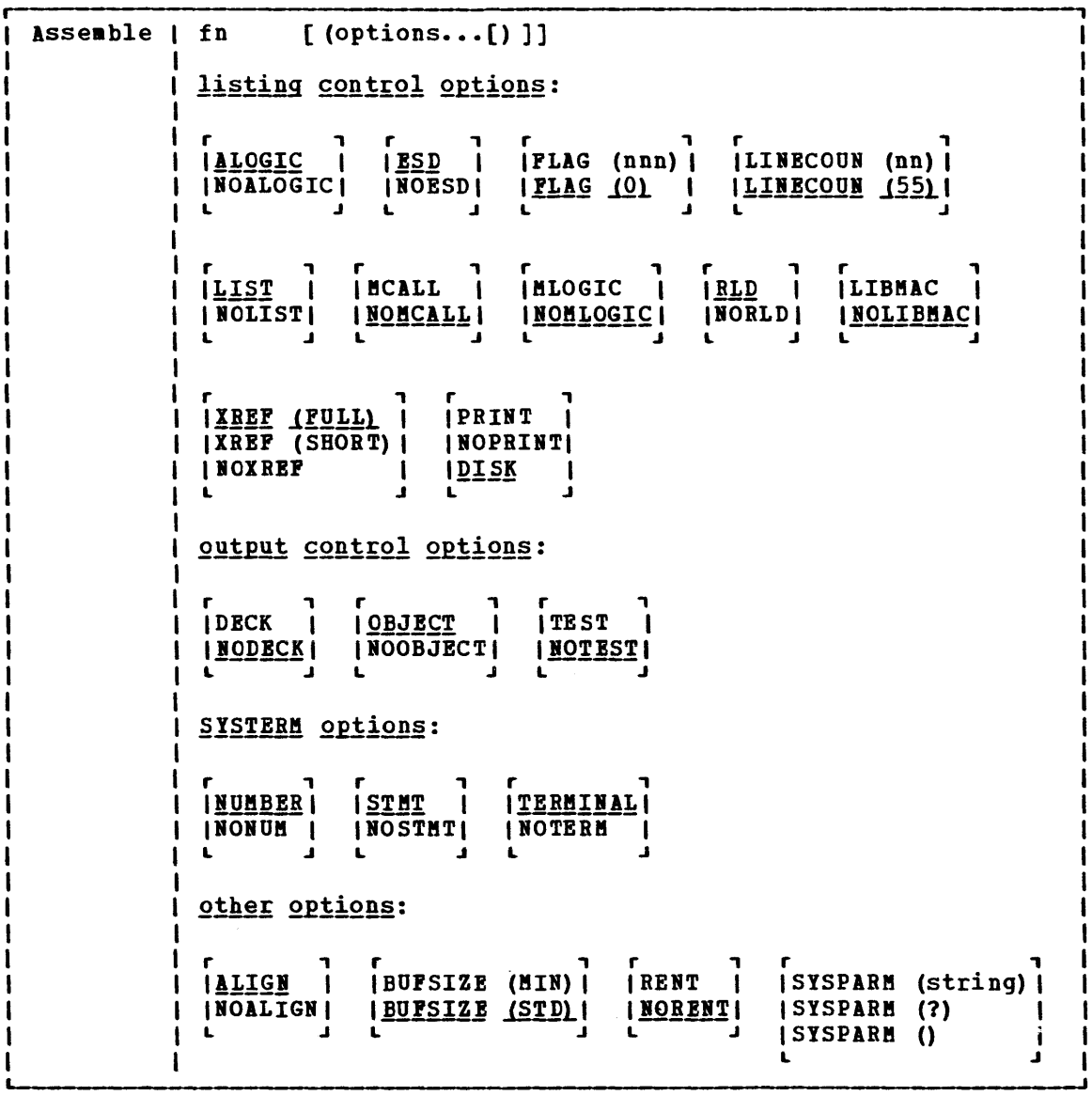
The disk located at virtual address cuu that is being accessed as a read/write disk is already currently accessed. The effect of the previous ACCESS command is canceled and the disk is released from the virtual machine configuration.

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 NOT
           BE ACCESSED  RC=28
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
DMSACC240S ERROR LOADING READ OS ROUTINE 'DMSROS TEXT'
```

ASSEMBLE

Use the ASSEMBLE command to invoke the Assembler to assemble the specified file. The Assembler processing and output are controlled by the options selected. The format of the ASSEMBLE command is:



where:

fn is the filename of the source file to be assembled. The file must have a filetype of ASSEMBLE and fixed-length, 80-character records.

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 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) LINECOUN (55)	nn specifies the number of lines to be listed per page.
LIST	produces an assembler listing.
NOLIST	does not produce an assembler listing. 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 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 printing 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 together with the OJBECT 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 specified in the TEXT FILEDEF statement. If this option is specified together 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.
TEST	includes the special source symbol table (SYM cards) in the object module.
NOTEST	Does not produce SYM cards.

SYSTERM OPTIONS: These 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 options below 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.

SYSPARM { (string) | () null string | (?) } — 'string' is the value assigned to the system variable symbol &SYSPARM. The varible (string) cannot be greater than 8 characters. If you wish to enter a string of more than 8 characters, use the SYSPARM (?) format. With the SYSPARM (?) form, CMS prompts you with the message:

```
ENTER SYSPARM:
```

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

OVERRIDING CMS FILE DEFAULTS: When you issue the ASSEMBLE command, there are default FILEDEF commands issued for assembler data sets. You may want to override these with explicit FILEDEF commands. The ddnames most likely to be overridden 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)

The default FILEDEF commands issued by the assembler for the 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
```

A FILEDEF command, issued for any of the above ddnames prior to invoking the assembler, overrides the default FILEDEF issued by the assembler. Assume that there is an assembler source file in card deck form which you want to assemble. If you have this card deck read into your virtual machine reader, you must issue an overriding FILEDEF command prior to assembling, that is, FILEDEF ASSEMBLE READER. Now you can invoke the assembler as follows:

```
ASSEMBLE SAMPLE (options . . . .
```

The name SAMPLE is used by the assembler as the filename for any TEXT or LISTING files produced by the assembler, provided a file SAMPLE ASSEMBLE does not exist on any accessed disk, in which case, the first file is erased.

Similarly, if you have a tape containing an assembler input file which you want to assemble, you must issue the following commands:

```
FILEDEF ASSEMBLE TAPn  (RECFM F LRECL 80 BLOCK 80
```

or, if the file is blocked,

```
FILEDEF ASSEMBLE TAPn  (RECFM FB LRECL 80 BLOCK 80*n
```

followed by

```
ASSEMBLE SAMPLE  (options . . . .
```

You can use OS data sets as CMS files by defining those data sets with the FILEDEF command. For example,

```
FILEDEF ASSEMBLE DISK MYDSET ASSEMBLE B4 DSN OS DATASET
```

where:

B4 is the mode of OS disk to be accessed.

OS.DATASET is the name of the OS data set to be used for input.

To assemble this, issue:

```
ASSEMBLE MYDSET
```

The same examples used here for input files can be applied to other ddnames. Care should be taken that any attributes specified for the file conform to the assembler expected attributes for the device, that is, PUNCH, LRECL 80 BLOCK 80, TERMINAL 132.

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.

CMSBATCH

You use the CMSBATCH command to invoke the CMS Batch Facility. The format of the CMSBATCH command is:

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

where:

sysname is the name of the saved system that the CMS Batch Facility loads for all subsequent batch jobs. This operand can only be used if the installation has set up the CMS Batch Facility as a saved system.

You issue this command immediately after loading CMS with the IPL command. You can find a complete description of the CMS Batch Facility in "Appendix C: Using the CMS Batch Facility" and in the VM/370: Operator's Guide.

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 disk files of fixed or variable length format and to display the contents of corresponding unlike records at the terminal. The format of the COMPARE command is:

```
| COMpare | fileid1 fileid2  [(COL mm-nn[)]]                          |
```

where:

fileid — is the file identification of the two files to be compared. All three identifiers (filename, filetype, and filemode) must be specified for each fileid.

Option

(COL mm-nn) (COL 1-lrecl) — defines any contiguous portion of the corresponding records for comparison. The comparison begins at position mm of each record in both files. The comparison proceeds up to and including position nn of each record in both files. If mm is not specified, the comparison starts with the first character of each record in both files. If nn is not specified, the default ending position is the last character of each record. The "-" is required and may not be preceded or followed by a blank. "lrecl" is the logical record length of the file.

If you want to stop the displaying of the dissimilar records, use the CMS immediate command HT.

Examples

```
COMPARE ABC XYZ A1 ABC MNO A1
```

Each record in file ABC XYZ A1 is compared with the corresponding record in file ABC MNO A1. Comparison begins at the first position of each record and proceeds for the entire length of the record. Records which do not match are displayed at the terminal.

```
COMPARE MYFILE ASSEMBLE A1 YOURFILE ASSEMBLE A1 (COL 10-72)
```

Positions 10 through 72 of each record in file MYFILE ASSEMBLE A1 are compared with corresponding positions of each record in file YOURFILE ASSEMBLE A1. Records in which these positions do not match are displayed at the terminal.

Responses

If corresponding records in each file do not match, the record from the first file is displayed, followed by the record from the second file.

Error Messages and Return Codes

```
DMSCMP002E FILE 'fn [ft [fm]]' NOT FOUND  RC=28
DMSCMP003E INVALID OPTION 'option'  RC=24
DMSCMP005E NO 'option' 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 'param' 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
```

COPYFILE

Use the COPYFILE command to copy data from specified input files to the output files according to conversions and specifications indicated by the options selected. The manner in which the file identifiers are entered determines whether one output file is created (single output mode) or multiple files are created (multiple output mode).

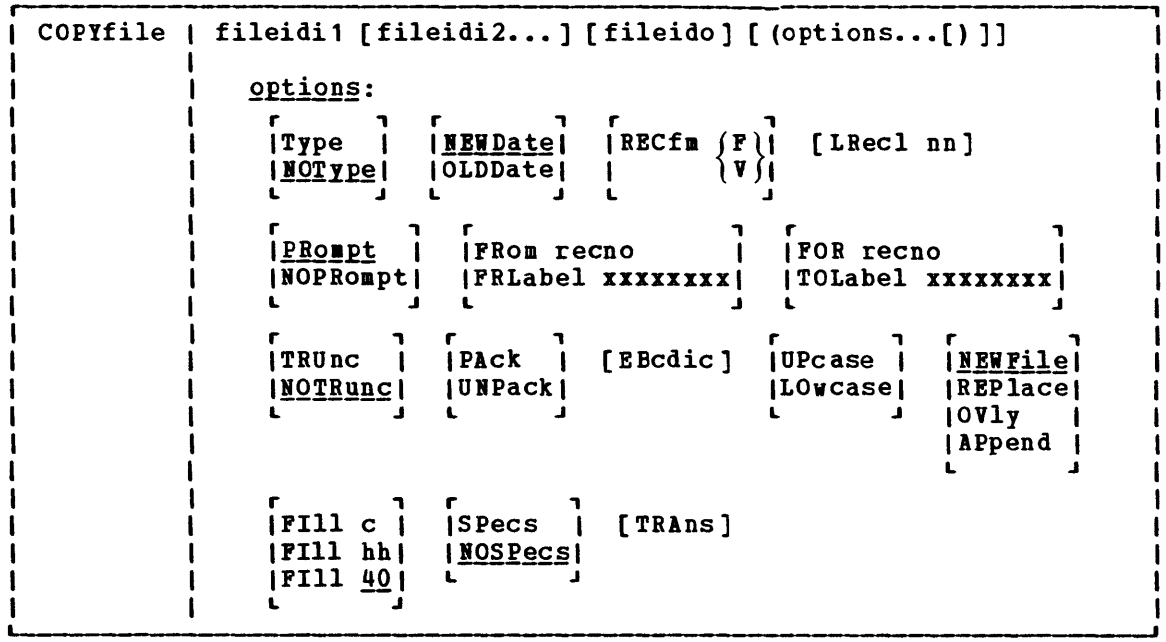
The COPYFILE command is used to:

- Copy one file to another
- Combine two or more files into a single output file
- Copy files into multiple output files
- Copy a file from one minidisk to another

With the COPYFILE command, you can:

- Display the names of files copied at the terminal.
- Replace the existing output files with the new output files.
- Change the record format and logical record length.
- Selectively copy records from the input file(s) based on either:
  - Record number
  - Label field of record
- Remove the trailing fill characters from each record.
- Compress an input file.
- Convert 026 key punch characters to corresponding 029 characters.
- Convert lowercase letters to uppercase and uppercase letters to lowercase.
- Overlay data in an existing file with selected data from another file.
- Append one file to another file.
- Move selected positions within each record to specified positions in the records of another file.
- Insert a specified character string or hexadecimal character into selected positions of each record in the output file.
- Perform character translations.

The format of the COPYFILE command is:



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. However, all three file identifiers of fileidi1 cannot be specified by asterisks.

fileidi2 is an additional input file(s). 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 for any of the file identifiers, indicating that it is the same as the corresponding identifier in fileid1.

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 fileid, fileido may be omitted, in which case it defaults to = = = (the input file represented by fileidi1 is replaced).

The following options are the most used options of the COPYFILE command. Minor variations of these options are described in the "Other Options" section.

Options

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

NOTYPE suppresses the displaying at the terminal the names of the files being copied.

NEWDATE uses the current date as the creation date of the new files.

OLDDATE uses the date on the first input file as the creation date of the new files.

RECFM { F / V } is the record format of the output files. If not specified, the output record format is the same as that of the input.

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

PROMPT displays the messages which 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 sequence which appears at the beginning of the first record to be copied from each input file. Up to eight characters may be specified. The character sequence may not contain embedded blanks.

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

TOLABEL xxxxxxxx
xxxxxxxx is a character sequence 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 characters may be specified. The character sequence may not contain embedded blanks.

Character Translation Options

TRUNC removes trailing blanks (or fill characters) when converting to RECFM V output file record format.

The RECFM and LRECL options can be used to specify the record format and logical record lengths of the output files being created.

There are two record formats, F (fixed) or V (variable). All records in a fixed record format file are the same size (for example, card image files, where all records are 80 bytes long). Variable record format files have records of varying sizes. SCRIPT files, where the length of each record is usually different, are variable record format files.

The logical record length (LRECL) applies only to fixed record format (F) files. For example, for card image files, the logical record length is 80.

If the RECFM or LRECL options are not specified, the record format and logical record length for the output file are the same as for the current or only input file. If the output file record format is V, then the LRECL option (if specified) is ignored.

When the output file record format is F, the input records are truncated or padded, as necessary, to the logical record length. When the output file record format is V, the input records are simply copied to the output file, unless the TRUNC option is specified. If TRUNC and RECFM V are specified, all blanks at the end of each record are removed before the record is written out.

For information on variations of this option, see the discussion of the FILL option under "Other Options."

NOTRUNC suppresses the removal of trailing blanks (or fill characters) when converting to RECFM V output file record format.

PACK converts repetitively occurring characters to compressed format. If the FILL option is not used, all occurrences of two or more blanks in the file are encoded as one character, and four or more occurrences of any other character in the file are encoded as three characters. If a FILL character is specified, that character replaces the blank as the special packing character, and blanks are treated as any other non-fill character.

Source files generally take up a great deal of disk space, because they contain many blanks. This is particularly wasteful in the case of source files which are seldom used.

Use the PACK option of the COPYFILE command to encode a file so that multiple blanks are represented as a single character, and multiple occurrences of other characters also produce space savings.

When the PACK option is used, the output file is in a format which can be decoded only by the UNPACK option of the COPYFILE command.

For example, the command:

```
COPYFILE * ASSEMBLE A1 (PACK
```

causes all ASSEMBLE files on the A-disk to be packed.

If you know that a particular character occurs a great many times in a file (and if this character occurs more often than the blank character), specify that character as the fill character for the file. Two or more occurrences of the specified fill character are encoded as one character. Then four or more consecutive occurrences of any other character (including the blank) are encoded as three characters.

However, not every file should be packed. In fact, if the file does not contain very many occurrences of multiple characters, it is possible for the packed file to be longer than the original file.

CAUTION: A file in packed format should not be modified in any way. If such a file is modified in any way, the UNPACK routines will be unable to reconstruct the original file. Packed files should never be combined or split.

UNPACK reverses the PACK operation.

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 be written to an output file to uppercase before the record is written out.

LOWCASE converts all uppercase characters in each record to be written to an output file to lowercase before the record is written out.

Other Options

In addition to the options already described, the COPYFILE command offers several variations:

NEWFILE checks that output files did not previously 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 be replaced by an input file of the same name. REPLACE is the default option when the output file identification is "= = =."

OVLY — overlays the data in an existing output file with data from the input file.

APPEND — appends the data from the input file to the end of a file specified by the output file identifiers. If no output file exists, one is created.

FILL c
FILL hh
FILL 40 — is the padding and truncation character or the principal packing character for the PACK option. The fill character may be specified by entering 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).

When the output file record format is F, the input records are truncated or padded, as necessary, to the logical record length. The padding and truncation character is usually a blank, but this default may be overridden if you specify a new padding and truncation character with the FILL option.

When RECFM V and TRUNC are specified, all blanks on the end of each record are removed before the record is written. To truncate some character other than blanks, use the FILL option.

The PACK routine treats the blank as a "special packing character". If desired, the FILL option may be used to change the "special packing character" to a non-blank character.

SPECS — requests a user specification list defining the manner in which data is to be copied.

If the SPECS option is used, COPYFILE issues the prompting message:

```
DMSCPY601R ENTER SPECIFICATION LIST
```

The keyboard unlocks, and you may type in the specification list.

The format of the specification list is:

```
source target [source target] ...
```

where source can be specified in several ways and target is a decimal number representing a position in the output record. Source can be specified as:

- A pair of columns of the input file, specified in the format "nn-mm", two decimal number values separated by a hyphen. This format causes the specified positions in the input file record to be copied to the output file. For each record that is copied, the value of mm (the ending record position) is compared to the length of the record. Whenever the specified ending record position exceeds the length of the record, the end of the record becomes the assumed ending position for that record. For example, the source specification "23-40" causes columns 23 through 40 of the input file record to be copied to the output file.

- A string of characters, delimited by nonalphameric characters. Such a specification causes the specified string of characters to be placed in the output file record. For example, a source specification of "/AbCd1234/" causes the string "AbCd1234" to be placed in each record of the output file. The letters in the string of characters are not automatically raised to uppercase; if you wish capital letters, you must type them in as such.

- A string of characters, specified by the letter "H" followed by an even number of hexadecimal digits entered using numeric characters and either upper or lower case alphabetic characters. For example, the specification, "hb7c1ff" causes the characters given by the hexadecimal values X'B7', X'C1', and X'FF' to be placed in the output file.

For example, consider the following specification list:

```
1-5 10 /ABC/ 3  /XYZ/ 20 H00 25
```

This specification list contains four sets of specifications, and causes data to be placed into the output record in the following manner: First, positions 1 through 5 of the input record are placed in positions 10 through 14 of the output record. Next, the characters "ABC" are placed in positions 3 through 5 of the output record. Next, the characters "XYZ" are placed in positions 20 through 22 of the output record. Finally, the character X'00' is placed in position 25 of the output record.

For variable length output files, the length of the output record is determined by the position of the rightmost byte of data that was placed in the output record. For fixed length output files, the record is padded or truncated to the logical record length.

Positions of the output record for which no data is specified are filled with blanks. Use the FILL option to specify another fill character. In the example shown above, positions 1-2, 6-9, 15-19 and 23-24 of the output file record contain blanks. In addition, if the output record format is F, positions 26 through to the end of the record contain blanks.

The SPECS option is particularly useful in conjunction with the OVLY option. If these options are used together, the specification list indicates the exact positions of the output file record to be overlaid. For example, if the specification list 1-10 20 is used with the OVLY option, then positions 1-10 of each input file record overlay positions 20-29 of each record of the existing output file, and the other positions of the output file records remain unchanged.

The specification list can be continued onto additional lines by typing '++' at the end of the first line. If these two characters are encountered when a source-specification is expected, all scanning of that line ceases, and the keyboard unlocks so that a new line may be entered.

NOSPECS indicates that no specification list is to be entered.

TRANS specifies that you are to be asked for a list of character translations to be made as the file is copied.

The TRANS option allows you to specify your own list of translations; it overrides any of the other three translation options (UPCASE, LOWCASE or EBCDIC).

When the TRANS option is specified, the COPYFILE command displays the prompting message:

```
DMSCPY602R ENTER TRANSLATION LIST
```

You may then enter the translation list.

The translation list consists of a series of pairs of characters, separated by blanks. Each character may be specified either by entering the character itself or by entering a two-digit hexadecimal equivalent (the latter is particularly useful for characters not available on your terminal keyboard).

For example, the translation list:

```
* - A f0 00 ff
```

specifies that the character '*' is to be translated to '-', the character 'A' is to be translated to X'F0', and the character X'00' is to be translated to X'FF'. If the preceding translation list is specified in conjunction with the LOWCASE option, then the translation 'A' to X'F0' overrides the LOWCASE translation, 'A' to 'a'.

The translation list can be continued onto additional lines by typing '++' as the last two characters of the present line. Whenever these two characters are encountered when the first character of a character pair is expected, all scanning of that line ceases, (for typewriter terminals, the keyboard unlocks), then a new line may be entered.

Incompatible Options

Figure 14 shows combinations of options which 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 can 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, TOLABEL, TRANS, TRUNC, UPCASE
UPCASE	PACK, UNPACK

Figure 14. COPYFILE Option Incompatibilities

Examples

```
COPYFILE OLD FILE A1 NEW FILE A1
```

Copies the file OLD FILE A1 to a new file named NEW FILE A1.

```
COPYFILE * FILE A1 BIG FILE A1
```

Copies all files with filetype of FILE and filemode of A1 and combines them into a single output file named BIG FILE A1. If the files A FILE A1, B FILE A1, and C FILE A1 exist, they are copied to BIG FILE A1 and are not erased.

```
COPYFILE OLD * B1 NEW = B1
```

Copies a group of files, each with a filename of OLD and a filemode of B1, to a new group of files each with a filename of NEW and a filemode of B1. If the files named OLD NAME B1, OLD BIRTHDA B1, and OLD ADDRESS B1 exist, they are copied to files named NEW NAME B1, NEW BIRTHDA B1, and NEW ADDRESS B1, respectively.

```
COPYFILE X Y A1 P Q A1 BIG FILE A1
```

Combines files X Y A1 and P Q A1 in a single output file named BIG FILE A1.

```
COPYFILE X * A1 = FILE =
```

Combines all files with a filename of X and a filemode of A1 in a single output file named X FILE A1.

```
COPYFILE X * A1 = T A1 BIG FILE A1
```

Combines all files having a filename X and filemode A1 with the file having a filename X, a filetype T, and filemode A1 to form a single file named BIG FILE A1. The command

```
COPYFILE X * A1 = T = BIG FILE =
```

produces the same result.

```
COPYFILE X * A1 P Q A1 BIG = A1
```

Combines each file with a filename of X and a filemode of A1 with the file P Q A1 to create a file named BIG * A1. One file is created for each existing file with a filename X and filemode A1.

```
COPYFILE X * A1 P = A1 BIG = A1
```

Combines each file with a filename of X and a filemode of A1 with a file with a filename of P, a filemode of A1 and a filetype corresponding to that of the first input file; to produce a file with a filename of BIG, a corresponding filetype, and a filemode of A1.

```
COPYFILE X * A1 P = = BIG = =
```

produces the same result.

```
COPYFILE BIGNAME A A1 XYZ= == =
```

Copies the file BIGNAME A A1 to a file named XYZBIGNA AA A1.

```
COPYFILE A B A1 C D A1 (FROM 10 FOR 25)
```

Copies 25 records from file A B A1 to file C D A1, beginning with the tenth record of A B A1.

```
COPYFILE OLD B A1 NEW B A1 (FRLABEL MYPROG TOLABEL FINIS)
```

Copies records from file OLD B A1 to NEW B A1, beginning with a record containing the characters 'MYPROG' and continues until a record containing the characters 'FINIS' is encountered.

```
COPYFILE * ASSEMBLE * (PACK
```

Converts all files with a filetype of ASSEMBLE to packed format.

```
COPYFILE MY FILE A1 YOUR FILE A1 (SPECS
```

Copies the file named MY FILE A1 to a new file named YOUR FILE A1 modified according to the specification list. The message

```
DMSCPY601R  ENTER SPECIFICATION LIST
```

is displayed.

For example, if you enter the following specification list:

```
1-5  10    /ABC/ 3   H00 1
```

Positions 1 through 5 of the input record are copied to positions 10 through 14 of the output record, the character string "ABC" is placed in positions 3 through 5 of the output record, and the character X'00' is placed in the first position of the output record. All other positions in the output record contain blanks unless a FILL character is specified, in which case the FILL character is placed in all unspecified positions.

```
COPYFILE OLD FORM A1 NEW FORM A1 (SPECS OVLY
```

Portions of the existing output file are to be overlaid by portions of the input record, or by character strings specified in a specification list.

```
COPYFILE XYZ TEXT C = = A
```

Copies a file from one virtual disk to another.

Responses

DMSCPY601R ENTER SPECIFICATION LIST:

This message requests the specification list which is to be entered in conjunction with the SPECS option.

DMSCPY602R ENTER TRANSLATION LIST:

This message requests the translation list which is to be entered in conjunction with the TRANS option.

DMSCPY721I COPY 'fn ft fm' {TO |APPEND| OVLY} 'fn ft fm' {OLD|NEW} FILE

This message appears in conjunction with the TYPE option. It indicates the names of the input file and output file.

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 'parm' 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 'hh'  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 permitted for your CP command privilege class. If this field is omitted, you are placed in CP mode and may enter CP commands without preceding each command with CP. To return to CMS issue the CP command BEGIN.

Responses

All responses are from the CP command which was issued.

Example:

```
CP TERMINAL LINESIZE 80
```

The TERMINAL command is passed to the control program for processing.

DDR

Use the DASD Dump Restore (DDR) service 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.

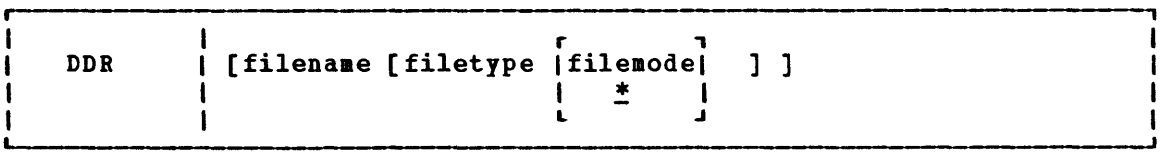
3. Copies data from one device to another of the same type. Data may be reordered, by cylinder, when copied from disk to disk. In order to copy one tape to another, the original tape must have been created by the DDR DUMP function.

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.

Note: To generate the VM/370 starter system from the distribution tape, the standalone RESTORE function must be used.

The format of the DDR command is:



where:

filename filetype [filemode] 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.

DDR CONTROL STATEMENTS

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 cylinder. Only

one type of function - dump, restore, or copy - may be performed in one execution, but up to 20 statements describing cylinder 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 certain control cards must be repeated. Only those statements needed to redefine the I/O devices are necessary for subsequent steps. All other I/O definitions 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.

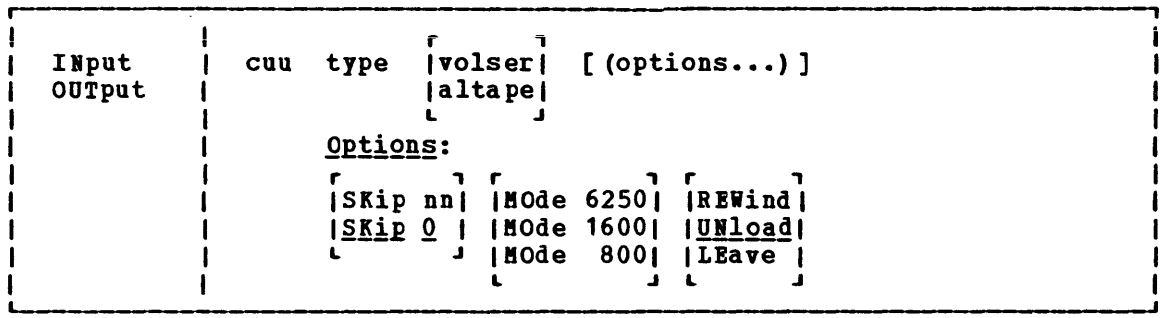
The PRINT and TYPE statements work differently from other DDR control statements in that they operate on only one data extent at a time. If the input is from a tape created by the dump function, it must be positioned at the header record for each step. The PRINT and TYPE statements have an implied output 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:



where:

INPUT indicates that the device described is an input device.

OUTPUT indicates that the device described is an output device.

cuu is the unit address of the device.

type is the device type (2314, 2319, 3330, 3330-11, 3340-35, 3340-70, 2305-1, 2305-2, 2400, 2420, or 3420) (no 7-track support for any tape devices). Specify a 3410 device as a 3420, a 3340-70F as a 3340-70, and a 3333 as a 3330.

Note: The DASD Dump Restore (DDR) program, running 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:

DMKDDR701E INVALID OPTION

However, if DDR runs standalone in a virtual machine, DDR uses DIAGNOSE 20 to perform the I/O operation if the device types agree and SIO if the device types do not agree.

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 MOUNT NEXT TAPE." After the new tape is mounted, DDR continues automatically.

Options

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

MODE 6250 causes all output tapes that are opened for the first time
MODE 1600 and at the load point to be written or read in the specified
MODE 800 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.

SYSPRINT Control Statement

Use the SYSPRINT control statement (in the standalone application 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 COpy REstore	{ cyl1 [To] [cyl2 [Reorder] [To] [cyl3]] CPvol ALL NUcleus }

where:

DUMP requests the program to move data from a direct access volume onto a magnetic tape or tapes. The data is moved cylinder by cylinder. Any number of cylinders may 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 or end-of-job trailer label. The end 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.

COPY requests the program to copy data from one device to another device of the same or equivalent type. Data may be recorded on a cylinder 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 as it was dumped from. 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, 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.

CPVOL specifies that cylinder 0 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 service program must have formatted them.

ALL specifies that the operation is to be performed on all cylinders.

NUCLEUS specifies that record 2 on cylinder 0, track 0 and the nucleus cylinders will be dumped, copied, or restored.

Restrictions:

1. Each track must contain a valid home address, containing the real cylinder and track location.

2. Record zero must not contain more than eight key and/or data characters.

3. Flagged tracks are treated just as any other track for all 2314, 2319, 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.

4. Flagged tracks for a 3330 device 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, 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 room on the first tape, it continues dumping onto the alternate device (181). While dumping, a map of the dumped cylinders is printed on unit 00F. 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 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.

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 or 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 TYpe	cyl1 [hh1 [rr1]] [To cyl2 [hh2 [rr2]]] [(options)] 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 1 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.

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.

Examples:

```
PRINT 0 TO 3
```

Prints all of the records from cylinders 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 15 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.

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.

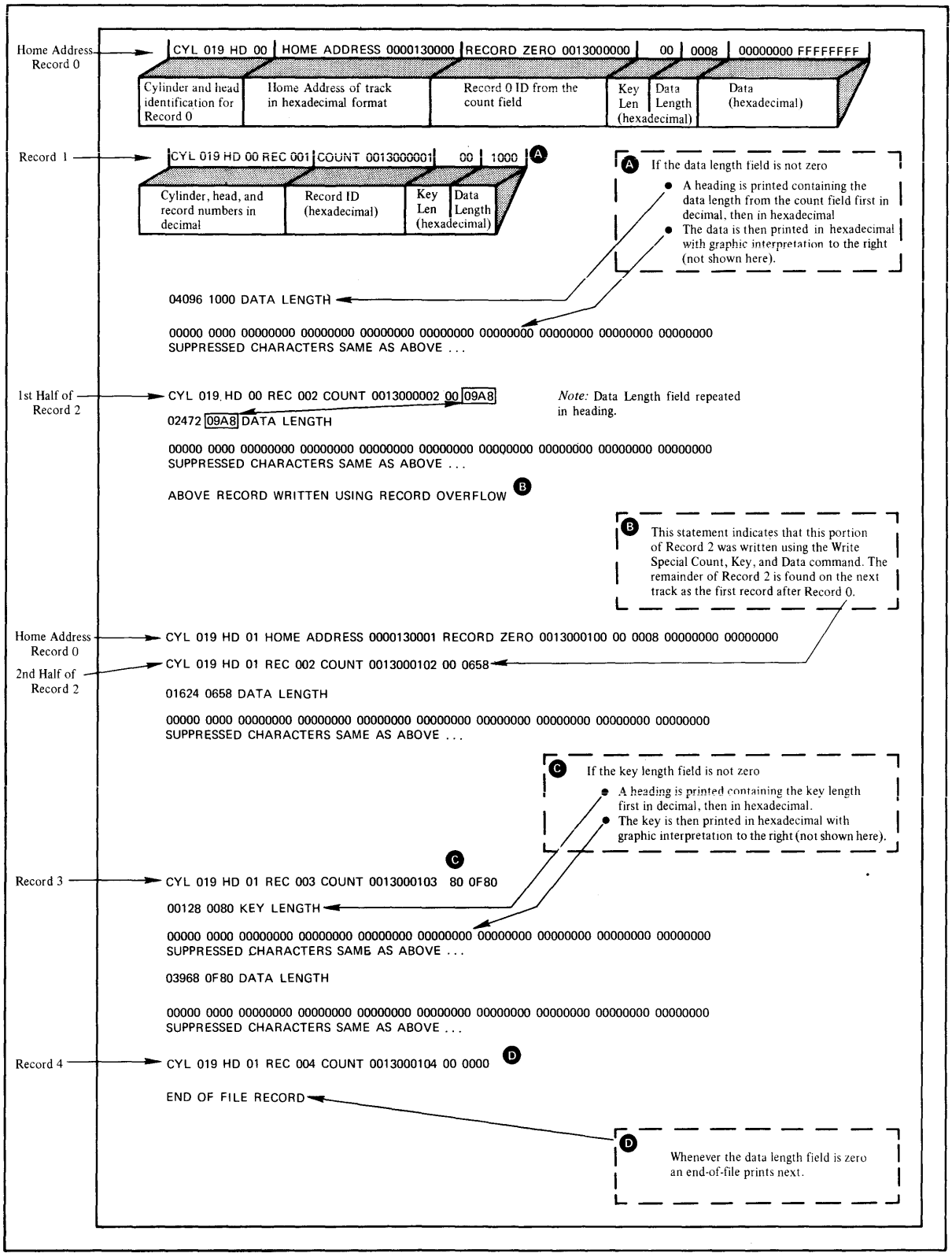


Figure 15. An Annotated Sample of Output from the TYPE and PRINT Functions of the DDR Program

```
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
ENTER:
```

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

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

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

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

DEBUG

Use the DEBUG command to acquire online facilities for debugging programs running under CMS and to acquire an entry in CMS for handling external interrupts, program interrupts, and unrecoverable errors. The subcommands which may be issued in the DEBUG environment allow you to examine and change the contents of certain control words and registers as well as portions of virtual storage. The facilities of DEBUG are made available when:

- The DEBUG command is issued
- An external interrupt occurs
- A break point (instruction address stop) is encountered during program execution; this causes a program interrupt.

Once the DEBUG environment has been entered due to any of the above circumstances, you are in the DEBUG environment. Only DEBUG subcommands are valid input in this environment.

When the DEBUG environment is entered, the contents of all general registers, the channel status word, and the channel address word are saved so they may be examined and changed before being restored when leaving the DEBUG environment. If DEBUG is entered via an interrupt, the old program status word for that interrupt is also saved. If DEBUG is the first command entered after an ABEND occurs, the contents of all general registers, the CSW, the CAW, and the old PSW are available from the time of the ABEND. The format of the DEBUG command is:

```
| DEBUG |                                                              |
```

Notes:

1. The CMS commands HB, HO, HT, RO, RT and SO are not recognized in the DEBUG environment.
2. The floating-point registers cannot be examined or changed in the DEBUG environment.

Responses

None.

DEBUG SUBCOMMANDS: For a complete description of the DEBUG subcommands refer to the VM/370: System Programmer's Guide. Figure 16 summarizes the DEBUG subcommands.

Subcommand Format	Function
BReak id {symbol / hexloc}	Stops program execution at the specified breakpoint.
CAW	Displays the contents of the Channel Address Word at the time DEBUG was entered.
CSW	Displays the contents of the Channel Status Word at the time DEBUG was entered.
DEFine symbol hexloc [n / 4]	Assigns a symbolic name to the virtual storage address.
DUmp [symbol1 / hexloc1] [symbol2 / hexloc2 / *] [ident]	Dumps the contents of specified virtual storage locations to the virtual spooled printer.
GO [symbol / hexloc]	Returns to the CMS environment and starts execution at the specified location.
GPR reg1 [reg2]	Displays the contents of the specified general registers.
HX	Returns to the CMS command environment.
ORigin [symbol / hexloc]	Specifies the base address to be added to locations specified in other DEBUG subcommands.
PSW	Displays the contents of the old Program Status Word.
RETurn	Exits from DEBUG environment to the CMS command environment.
SET {CAW hexinfo / CSW hexinfo [hexinfo] / PSW hexinfo [hexinfo] / GPR reg hexinfo [hexinfo]}	Changes the contents of specified locations or registers.
STore {symbol / hexloc} hexinfo	Stores information in a specified virtual storage location.
X {symbol / hexloc} [n / 4]	Examines virtual storage locations.

Figure 16. Summary of DEBUG Subcommands

DISK

Use the DISK command to:

- Punch 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 indicates that the specified file (fn ft fm) is to be punched. 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 indicates that one or more card files are to be read from the spooled card reader and written as CMS files on disk. The DISK LOAD operation reads a card deck consisting of any number of logical decks previously punched by DISK DUMP. The file designations are obtained from the card stream. If a file exists with the same designation as one of those in the card stream, it is erased and replaced. DISK LOAD loads files onto the primary read/write disk.

Examples

```
DISK DUMP MYFILE OLD A1
```

The specified file, MYFILE OLD A1, is written to the virtual spooled punch and followed by an end-of-file card.

```
DISK LOAD
```

All files which were previously read into the virtual card reader are to be loaded onto disk; each file must be followed by an end-of-file card as created by the DISK DUMP function.

Responses

There is no response to the DISK DUMP command. The file identifiers of each file loaded is the response for the DISK LOAD command:

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

Other Messages and Return Codes

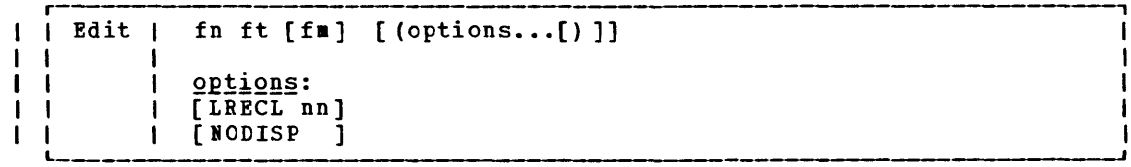
```
DMSDSK002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSDSK009E COLUMN 'col' EXCEEDS RECORD LENGTH  RC=24
DMSDSK014E INVALID FUNCTION 'function'  RC=24
DMSDSK037E DISK 'mode' IS READ/ONLY  RC=36
DMSDSK047E NO FUNCTION SPECIFIED  RC=24
DMSDSK048E INVALID MODE 'mode'  RC=24
DMSDSK054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSDSK062E INVALID * IN FILEID ['fn ft fm']  RC=20
DMSDSK070E INVALID PARAMETER 'param'  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
```

EDIT

Use the CMS Editor to:

- Create, from the terminal, sequential files consisting of either fixed or variable length records.
- Provide, on the basis of certain known filetypes, automatic selection of record length, record format, logical tab settings, serialization, linemode, and lowercase to uppercase translation.
- Add, delete, or change any part of a CMS file.
- Extract all or part of a CMS file to create a new file, or to embed it in another file.
- Allow searching and changing of portions of the file through context-directed searches or by using a specific line number.
- Receive automatic prompting with line numbers.
- Allow any or all of the file to be displayed at your terminal.
- Provide an interface to the CMS EXEC interpreter to provide a macro facility. Refer to the VM/370: EDIT Guide for a complete description of the macro facility.

The format of the EDIT command is:



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 a new file created, and after the input environment is entered, information entered by you becomes input to that file. If a file with the same filename and filetype does exist, the EDIT environment is entered, enabling you to issue EDIT subcommands and to modify the specified file. Both the filename and filetype must be specified.

fm — is the filemode of the file to be created or edited. If the filemode is specified, EDIT searches only the indicated disk. If the filemode is specified as an asterisk, EDIT searches all disks for the existence of that file. If the filemode is not specified, only the primary disk and its extensions are searched. If the file is found, its mode is saved, and EDIT later writes the altered file back to that same disk. If an existing file is not found, the newly created file is placed on the disk specified by the filemode or on the primary disk.

Option

LRECL nn — is the record length of the file to be created or edited. If the record length is not specified, the following default values are assumed:

Editing Existing Files

Existing record length is kept regardless of format.

Creating New Files

Variable format:
- Filetype LISTING 121 maximum
- Filetype SCRIPT: 132 maximum
- Filetype FREEFORT: 81 maximum

Fixed format:
- All filetypes: 80

The maximum record length supported by the Editor is 160 characters.

NODISP — forces a display terminal into LINE (typewriter) mode. When the NODISP option is in effect, all subcommands that are solely for control of display terminals (BACKWARD, CHANGE with no operands, FORWARD, and SCROLL(UP)) are made invalid for the EDIT session.

This option is used when an EDIT session is initialized from a local 3270 terminal and the typewriter mode is desired. The default mode of operation for an EDIT session is DISPLAY for a local 3270 and LINE for a remote 3270.

The mode of operation for either terminal can be changed during the session via the FORMAT subcommand and the DISPLAY or LINE operand, unless the NODISP option is in effect.

EDIT Subcommands

Refer to the VM/370: EDIT Guide for a functional description of each of the EDIT subcommands. The formats are given in Figure 17 for reference only.

Subcommand Format	Function
ALter {parm1} {parm2} [n | * | 1] [G | *]	Scans the next n records of the file, altering the specified parameter, either once in each line or for all occurrences in the line.
AUTOsave { n | OFF }	Automatically saves the file on disk after the indicated number of lines have been processed.
BAckward [1 | n]	Points the current line pointer to a line above the line currently pointed to. (For display terminals.)
Bottom	Makes the last line of the file the current line.
CASE [U | M]	Indicates whether translation to uppercase is to be done, or displays the current status.
Change /string1/string2[/ [n | 1 | *] [G | *]]	Changes string1 to string2 for n records or to EOF, either for the first occurrence in each line or for all occurrences.

Figure 17. Summary of EDIT Subcommands and Macros (Part 1 of 5)

Subcommand Format	Function
CMS	Enters CMS subset command mode.
DELete [n | 1 | *]	Deletes n lines or to the end of the file (*).
DOwn [n | 1]	Points to the nth line from the current line.
DString /string [/]	deletes all lines from the current line down to the line containing the indicated string.
FILE [fn [ft [fm]]]	Saves the file being edited on disk or changes its identifiers. Returns to CMS.
Find [line]	Searches the file for the given line.
FMode [fm]	Resets or displays the filemode.
FName [fn]	Resets or displays the filename.
FORMat {DISPLAY | LINE}	Switches the 3270 terminal between DISPLAY mode and LINE mode.
FOrward [1 | n]	Points the current line pointer to a line currently pointed to. (For display terminals.
Getfile fn [ft | *] [fm | *] [m | 1] [n | *]	Inserts some, or all, of the given file following the current line.
IMAGE [ON | OFF | CANON]	Expands text into line images or displays current settings.
Input [line]	Inserts 'line' in the file or enters INPUT mode.
LINEmode [Left | Right | OFF]	Sets or displays current line setting.

Figure 17. Summary of EDIT Subcommands and Macros (Part 2 of 5)

Subcommand Format	Function
Locate /string [/]	Scans file from next line for first occurrence of 'string'.
LONG	Enters LONG error message mode.
Next [n | 1]	Points to the nth line down from the current line.
Overlay line	Replaces all or part of the current line.
PREserve	Saves current mode settings.
PROMPT [n]	Sets or displays line number increment. Initial setting is 10.
QUIT	Terminates EDIT session with no updates since last 'SAVE'.
RECfm [F | V]	Sets or displays record format for subsequent files.
RENum [strtno [10] [incrno [strtno]]]	Recomputes line numbers for VSBASIC and FREEFORT source files.
REPEAT [n | 1 | *]	Executes the following OVERLAY subcommand n times.
Replace [line]	Replaces the current line with 'line' or deletes the current line and enters INPUT mode (if no data line is specified).
REStore	Restores mode settings to values last preserved.
RETURN	Returns to EDIT environment from CMS subset.
{REUSE | =} [subcommand]	Stacks (LIFO) the last EDIT subcommand that does not start with REUSE or the question mark (?) and then executes any given EDIT subcommand.

Figure 17. Summary of EDIT Subcommands and Macros (Part 3 of 5)

Subcommand Format	Function
SAVE [fn [ft [fm]]]	Saves the file on disk and stays in EDIT environment.
S[croll][Up] [*] [n] [1]	Display a number of lines above or below the current line. (For display terminals.)
SERial { OFF / ON / ALL / seq [incr] [10] }	Turns serialization on or off in columns 73 through 80.
SHORT	Enters SHORT error message mode.
STACK [n] [1] [subcommand]	Stacks n lines, beginning with current line, in terminal input buffer.
TABSet {n1 n2 ... nx}	Sets the given tabs.
TOP	Points to the beginning of the file.
TRUNC [n] [*]	Sets or displays the column of truncation. An asterisk (*) means end of logical record.
Type [m] [1] [*] [n] [*]	Displays m lines beginning with the current line or the beginning of the file.
Up [n] [1]	Points to the line 'n' lines above the current line.
Verify [ON] [OFF] [startcol] [1] [endcol] [*]	Sets, displays, or resets verify mode. The asterisk (*) means end of logical record.
{X} {Y} [subcommand] [n] [1]	Assigns to X or Y the given EDIT subcommand or executes the previously assigned subcommand n times.
Zone [m] [1] [*] [n] [*]	Sets or displays the columns between which editing is to take place.

Figure 17. Summary of EDIT Subcommands and Macros (Part 4 of 5)

Subcommand Format	Function
?	Displays the last EDIT subcommand which did not begin with REUSE or a question mark (?).
nnnnn [text]	Locates the line specified by the given line number and inserts text if given.
$DUP [n | 1]	Duplicates the current line n times. $DUP is an EDIT macro.
$MOVE n { Up m | Down m | TO label }	Moves n lines up or down m lines. $MOVE is an EDIT macro.

Figure 17. Summary of EDIT Subcommands and Macros (Part 5 of 5)

Responses

NEW FILE:

The specified file does not exist.

EDIT:

The EDIT environment is entered. The logical tab settings may be either those defined by the user or those assumed according to the filetype. An EDIT subcommand may now be issued.

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.

?EDIT: line

"line" was entered in the EDIT environment and is an invalid EDIT subcommand or macro. This message appears while in LONG error message mode.

¬

An invalid subcommand was entered in the EDIT environment while in SHORT error message mode.

¬$

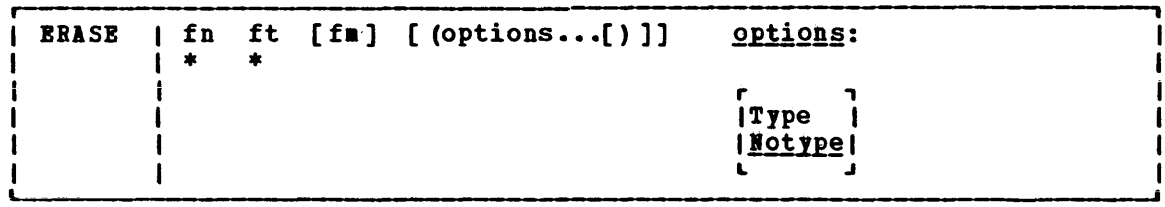
An invalid EDIT macro was entered while in SHORT error message mode.

Other Messages and Return Codes

```
DMSEDI003E INVALID OPTION 'option'  RC=24
DMSEDI024E FILE 'fn ft fm' ALREADY EXISTS  RC=28
DMSEDI029E INVALID PARAMETER 'param' IN THE OPTION 'LRECL' FIELD  RC=24
DMSEDI044E RECORD LENGTH EXCEEDS ALLOWABLE MAXIMUM  RC=88
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
DMSEDI132S FILE 'fn ft fm' TOO LARGE  RC=88
```

ERASE

Use the ERASE command to delete a file or a related group of files from a read/write disk. The file to be deleted must not be on a read-only disk. The format of the ERASE command is:



where:

fn is the filename of the files to be erased. An asterisk may be coded in this position to indicate that all filenames are to be used. This field must be specified, either with a name or an asterisk.

ft is the filetype of the files to be erased. An asterisk may be coded in this position to indicate 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, the primary read/write disk is searched for the files to be erased.

Note: If asterisk is specified for filename and filetype then filemode must be specified. The filemode must include both a mode letter and number.

Options

TYPE displays at the terminal the file identifier for each file erased.

NOTYPE file identifiers are not displayed at the terminal.

Example

```
ERASE OLDFILE TEMP (TYPE)
```

The file with the identifier OLDFILE TEMP, if located on the primary read/write disk, is erased. Its file identifier is displayed at the terminal.

Responses

fn ft fm

If the TYPE option is specified, the file identifier for each file erased is displayed.

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=24
DMSERS048E INVALID MODE 'mode'  RC=24
DMSERS054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSERS069E DISK 'mode(cuu)' NOT ACCESSED  RC=36
DMSERS070E INVALID PARAMETER 'param'  RC=24
DMSERS071E ERASE * * [*] NOT ALLOWED  RC=24
```

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, the DMSERS002E FILE fn ft fm NOT FOUND error message is not issued.

EXEC

Use the EXEC command if you want to be able to execute one or more CMS commands or EXEC control statements contained in a specified file by issuing a single command. If this command is entered from the CMS command mode but not nested within another EXEC procedure, the initial word 'EXEC' may be omitted. The format of the EXEC command is:

```
| EXec | fn [args...] |
```

where:

fn is the filename of a file containing one or more CMS commands to be executed. The filetype of the file must be EXEC and the file must be fixed format 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 a logical record length of 80 characters. Each EXEC file can contain a maximum of 4096 lines.

args are the arguments to replace the numeric variables in the EXEC file specified. Within an EXEC file, up to thirty symbolic variables can be used (each one indicated by an ampersand (&) followed by an integer ranging from one to thirty) to indicate values which are to be replaced when the EXEC file is executed. The arguments are assigned to symbolic variables in the order in which they appear in the argument list. For example, each time an &1 appears in an EXEC line, the first argument specified with the EXEC command temporarily replaces the &1, the second argument specified with the EXEC command replaces &2, and so on, to argument N of the EXEC command.

If the percent sign (%) is used in place of an argument, the corresponding variable (&N) is ignored in all the commands which refer to that variable. If the specified EXEC file contains more variables than arguments given with the EXEC command, the higher numbered variables are assumed to be missing, and are ignored when the commands are executed.

EXEC CONTROL STATEMENTS

Control statements begin with a control word, which is usually followed by a list of tokens, and in some cases by additional lines of data. Figures 18 and 19 list the control statements and their functions and the EXEC built-in functions. Refer to the VM/370: EXEC User's Guide for detailed information on how to use EXEC.

Control Statement	Function
&variable = ae	Assigns the value of ae to the symbol specified by &variable; ae is an algebraic expression; the equal sign must be preceded and followed by a blank.
&ARGS [arg1 [arg2 ...]]	Redefines the arguments &1, &2, ... with the value of 'arg1', 'arg2', ..., and resets the variable &INDEX.
&BEGPUNCH [ALL] line1 line2 . . &END	Punches 'line1', 'line2', ... into the card punch, without tokenizing them.
&BEGSTACK [FIFO|LIFO] [ALL] line1 line2 . . . &END	Stacks 'line1', 'line2', ..., in the console input buffer without tokenizing them.
&BEGTYPE [ALL] line1 line2 . . . &END	Displays 'line1', 'line2', ... at the console, without tokenizing them.
&CONTINUE	Provides a branch address for &ERROR, &GOTO, and other conditional branching statements.
&CONTROL [OFF|ERROR|CMS|ALL] [TIME|NOTIME] [PACK|UNPACK] [MSG|NOMSG]	Sets, until further notice, the characteristics of the summary of execution, which is automatically printed at the console.
&ERROR action	Executes 'action' following any CMS command which yields an error return code (that is, a return code which is not zero). 'action' can be any executable statement.

Figure 18. Summary of EXEC Control Statements (Part 1 of 3)

Control Statement	Function
&EXIT [returncode] (0)	Exits from the EXEC file with the given return code.
&GOTO {TOP / linenumber / label}	Transfers control to the top of the EXEC file, to the given line, or to the line starting with 'label'.
&If {token1 / &$ / &*} {EQ / NE / LT / LE / GT / GE} {token2 / &$ / &*} executable statement	Executes the 'executable statement' if the condition is satisfied.
&LOOP {n / label} {m / condition}	Loops through the following n lines, or down to (and including) the line starting with 'label', for m times, or until 'condition' is satisfied.
&PUNCH token1 [token2 ...]	Punches a card containing token1, token2, ...
&READ [n / 1 / ARGS / VARS var1 [var2 ...]]	Reads the next n lines from the terminal and treats them as if they had been in the EXEC file; or reads a line, assigns the tokens in it to the arguments &1, &2, ..., and resets &INDEX to the number of arguments thus set; or reads a single line and assigns the tokens in it to the variables 'var1', 'var2', ...
&SKIP [n / 1]	If n > 0, skips the next n lines of the EXEC file. If n < 0, transfers control to the line which is n lines above the current line. If n = 0, transfers control to the next line.
&SPACE [n / 1]	Displays n blank lines at the terminal.
&STACK [FIFO / LIFO] [token1 [token2 ...]]	Stacks a line in the terminal input buffer containing 'token1', 'token2', ..., or stacks a null line if the tokens are absent.

Figure 18. Summary of EXEC Control Statements (Part 2 of 3)

Control Statement	Function
&TIME [ON | OFF | RESET | TYPE]	Displays timing information.
&TYPE token1 [token2...]	Prints at the terminal a line containing 'token1,' 'token2,' ...

Figure 18. Summary of EXEC Control Statements (Part 3 of 3)

EXEC Built-in Functions

An EXEC built-in function consists of the name of the function and, usually, a list of arguments. Built-in function names are EXEC keywords, and start with an ampersand. With the exception of &LITERAL, they are recognized only if they appear as the token following the equal sign of an assignment statement. Figure 19 lists the EXEC built-in functions with their format and an explanation.

Built-in Function	Explanation
&CONCAT token1 [token2 ...]	Concatenates 'token1', 'token2', ..., into a single token, with a maximum length of eight.
&DATATYPE token	Has the value NUM or CHAR, depending on the data.
&Length token	Gives the number of nonblank characters in 'token'.
&LIteral token	Uses the literal value of 'token', without substitution for any EXEC variable which may appear in it.
&SUBSTR token i [j]	Extracts that part of 'token' which starts at character "i", with length "j"; or which starts at character "i" and runs to the end of the token.

Figure 19. Summary of EXEC Built-in Functions

Responses

As each CMS command in the EXEC file is processed, it is displayed at the terminal along with any nonzero return code. The &CONTROL command can be used to augment or reduce the amount of displaying done during execution.

If the EXEC interpreter finds an error, it displays the message:

```
ERROR IN FILE 'fn EXEC fm', LINE n, 'description of error'
```

where "description of error" is one of the following conditions with its appropriate return code:

Return Code	Description
(802)	&SKIP OR &GOTO ERROR
(804)	TOO MANY ARGUMENTS
(805)	MAX DEPTH OF LOOP NESTING EXCEEDED
(806)	DISK OR TERMINAL READ ERROR
(807)	INVALID SYNTAX
(808)	INVALID FORM OF CONDITION
(809)	INVALID ASSIGNMENT
(810)	MISUSE OF SPECIAL VARIABLE
(811)	ERROR IN &ERROR ACTION
(812)	CONVERSION ERROR
(813)	TOO MANY TOKENS IN STATEMENT
(814)	MISUSE OF BUILT-IN FUNCTION
(815)	EOF FOUND IN LOOP
(816)	INVALID CONTROL WORD

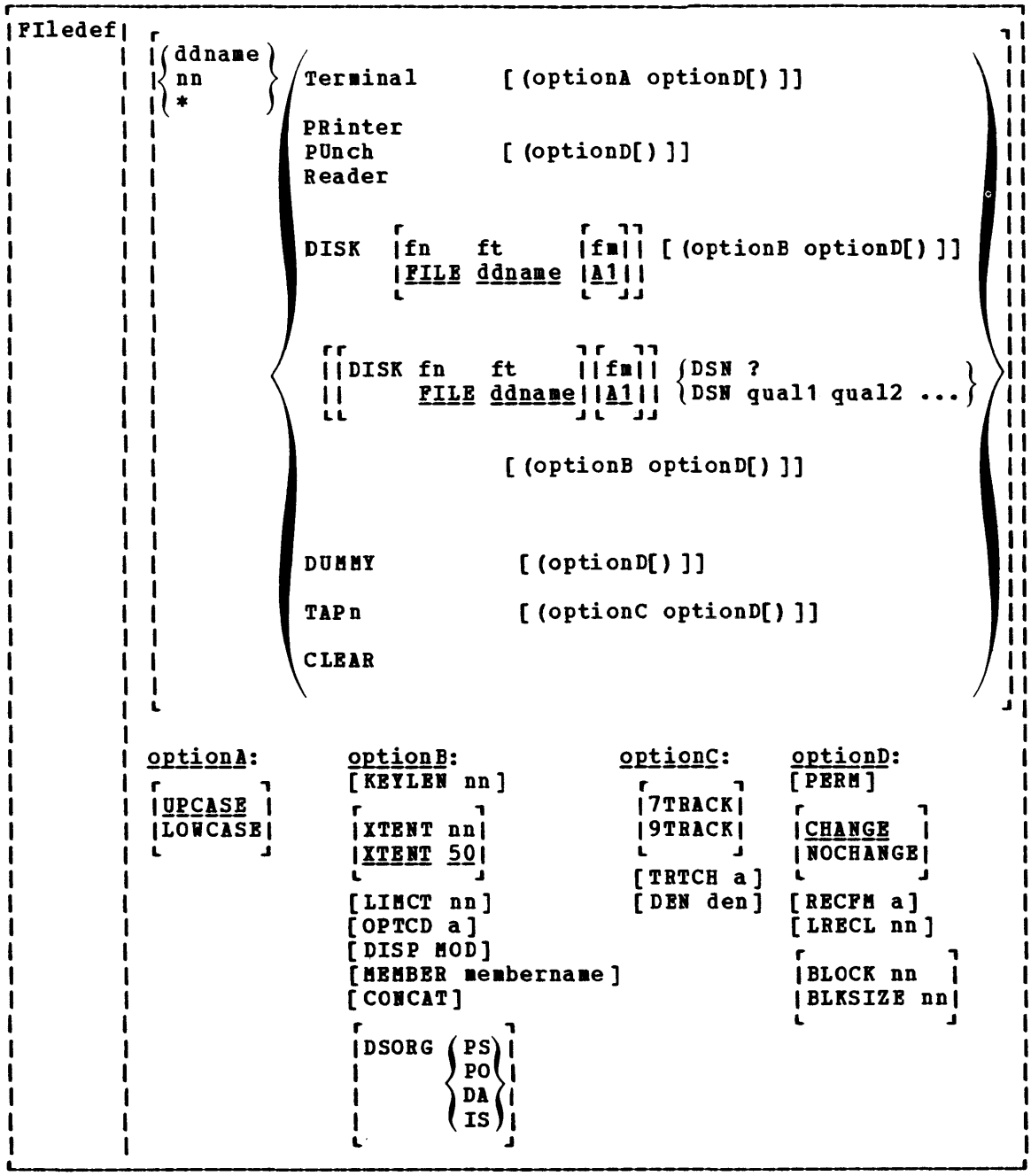
Error Messages and Return Codes

```
DMSEXC001E NO FILENAME SPECIFIED  RC=24
DMSEXC002E FILE 'fn ft' NOT FOUND RC=28
DMSEXC034E FILE 'fn ft fm' IS NOT FIXED LENGTH  RC=32
```

FILEDEF

Use the FILEDEF command to define an OS ddname and to relate that ddname to a device on your virtual machine. If the device is a disk, FILEDEF assigns a CMS file identification (that is, fn ft fm) and, if the disk is an OS disk, an OS data set name; or, if the disk is a DOS disk, a DOS file-id. Also, using FILEDEF, you can specify OS DCB parameters that describe the ddname just as they would on the OS Job Control Language Data Definition statement. In general, FILEDEF is used to define ddnames for programs written using the language processors supported by VM/370. You can find usage information on FILEDEF in "Using OS Programs and Macros Under CMS" in Section 4.

The format of the FILEDEF command is:



where:

ddname
nn
*
: is the name by which the file is referred to in your program. If a number nn is specified, it is translated to a FORTRAN[1] data definition name of FTnnF001. If the CLEAR operand is specified, ddname may be specified as an asterisk (*) to indicate that all file definitions not entered with the PERM option are to be removed.

Devices

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

PRINTER
: is the spooled printer.

PUNCH
: is the spooled punch.

READER
: is the spooled card reader (card reader I/O should 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 the section that follows, "Using the FILEDEF DISK Operand."

DUMMY
: indicates that no real I/O is to take place for a disk data set.

TAPn
: is a magnetic tape. The symbolic number of the tape drive, nn, can be 1, 2, 3, or 4, representing virtual units 181, 182, 183, and 184 respectively.

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.

If no operands are entered with the command, a list of current filetypes is displayed at the terminal including the ddname, device type, and, if device type is DISK, the filename, filetype, and filemode. Also, if the | FILEDEF is for an OS data set, the data set name is displayed; or, if | the FILEDEF is for a DOS file, the file-id is displayed.

Options

Whenever an option is specified that is invalid for a particular device type, an error message is issued. Figure 20 shows valid options for each device type.

---

[1]The FORTRAN processors are Program Products.

Option	Unit Record 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
KEYLEN				X
LIMCT				X[2]
LOWCASE, UPCASE		X		
LRECL	X	X	X	X
MEMBER				X
OPTCD				X[2]
PERM	X	X	X	X
RECFM	X	X	X	X
TRTCH			X[3]	
XTENT				X
7TRACK, 9TRACK			X	

[1]No options may be necessary but all disk options are accepted.
[2]This option is used for BDAM files.
[3]This option is for 7-track tapes only.

Figure 20. Valid File Characteristics for Each Device Type for the FILEDEF Command

UPCASE translates all terminal input data to uppercase.

LOWCASE retains all terminal input data as typed in.

KEYLEN nn is the size (nn) of the key (in bytes).

XTENT nn is the number of records (nn) in the extent for the file. The default is 50.

LIMCT nn is the maximum number of extra tracks or blocks (nn) to be searched. This option is used for BDAM files.

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

DISP MOD positions the read/write pointer after the last record in the disk file.

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 concatenate OS data sets with each other and with CMS files. You must use the CONCAT option when you are defining an OS data set referred to by a later GLOBAL command (CONCAT is not necessary for CMS files referred to in a later GLOBAL command.)

CONCAT should be used only for macro libraries to be specified in a later GLOBAL command. You must specify CONCAT with all of the OS data sets you define, including the first.

When you use CONCAT in a FILEDEF command, you can specify more than one FILEDEF for a particular ddname.

Also, when you use CONCAT, the DCB parameters used are those associated with the first filename you specify in a GLOBAL command for a macro library.

DSORG {PS / PO / DA / IS} is the data set organization: physical sequential (PS), partitioned (PO), direct access (DA), or indexed sequential (IS).

[7TRACK | 9TRACK] is the tape setting.

TRTCH a is the tape recording technique. Use the following chart to determine the value of a:

a	Parity	Converter	Translator
0	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 bpi (bits per inch). If 200 or 556 are specified, 7TRACK is assumed. If 800, 1600, or 6250 are specified 9TRACK is assumed.

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 exists for a ddname and a FILEDEF 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 nn — is the logical record length (nn) of the file, in bytes. LRECL should not exceed 32,767 bytes because of OS restrictions.

BLOCK nn
BLKSIZE nn — is the logical block size (nn) of the file, in bytes. BLOCK should not exceed 32,767 bytes because of OS restrictions.

If a CMS file is fixed and has 80-byte CMS records, you should specify RECFM FB BLOCK 800 LRECL 80 and a filemode number of 1. (BLOCK can also be expressed as 80*10.) There can be significant performance improvement for CMS fixed files if the block size is a multiple of 800.

Notes:

1. 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 on a terminal command. The option, AUXPROC addr, allows an auxiliary processing routine to receive control during I/O operations.

2. DOS files do not contain BLKSIZE, LRECL, or RECFM specifications. These options must be specified by a FILEDEF command or DCB statement. Otherwise the defaults, BLKSIZE=32760 and RECFM=U, are assumed. LRECL is not used for RECFM=U files.

3. If V or VS is specified for RECFM, LRECL must be at least 4 bytes less than BLKSIZE. BLKSIZE must be specified if LRECL is specified. The FILEDEF command does not provide default values for LRECL and BLOCKSIZE. However, the following chart describes the results of specifying BLOCKSIZE and LRECL.

---

[1] FB and VB should not be used with TERMINAL or READER devices.

[2] A and M may be used in conjunction with any of the valid RECFM settings (for example, FA, FBA, VA, VBA, etc.)

BLKSIZE	LRECL	Results
not specified	not specified	If the input file exists on disk, the item length becomes the BLKSIZE (or BLKSIZE +4 for variable length records).
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 record).
specified	specified	The values specified are used.

Using the FILEDEF DISK Operand

There are two general forms for specifying the FILEDEF DISK operand. If you specify the first form:

```
DISK fn ft [fm]
```

fn and ft (filename and filetype) are assumed to be a CMS fileid. If the filemode is for an OS disk, fn and ft are assumed to be the only two qualifiers of an OS data set name. In this form, the filemode you specify must always match the access mode of the disk on which the data set resides. The default values for fn ft fm are FILE ddname A1.

You cannot use this form unless the OS data set name or DOS file-id conforms to the OS naming convention (one- to eight-byte qualifiers separated by periods, to a maximum of 44 chracters, 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 data set name or 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 data set name or file-id is TEST.SAMPLE, then you enter:

```
FILEDEF MINE DISK TEST SAMPLE B1
```

The second form of the DISK operand is only for use 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 specify a DSN operand that corresponds to the DSN parameter on the DD card describing an OS data set or DOS file. There are three ways you can specify this form:

1. FILEDEF ddname DISK fn ft fm DSN qual1 [qual2...]

   If you use this form, the FILEDEF command associates the 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 coding the filename and filetype. If you omit DISK, filename, filetype, and filemode, the default values are FILE ddname A1.

2. FILEDEF ddname DSN ?

   This form of the FILEDEF command allows you to specify the DSN 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 filename and filetype associated with the OS data set name or DOS file-id. The filemode for this form is always the default, A1.

   The interactive DSN operand works this way: when you enter DSN ?, CMS requests that you enter the OS data set name or DOS file-id exactly as it appears in the data set or file. Do not 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.

3. FILEDEF ddname B1 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 one- to eight-byte qualifiers separated by periods, to a maximum of 44 characters, including periods.) Again, the default value for the filename is FILE and for filetype, the default value is the name associated with the OS data set name or DOS file-id. The filemode for this form is B1 as you specified it on the command. 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.

Note: When the FILEDEF command is used to define a ddname, that definition remains in effect until another FILEDEF command (with the CLEAR option) is issued or until the system clears the definition. The system clears FILEDEF definitions if:

- Any of the CMS language processors are executed. The language processors always issue, at their completion, the command FILEDEF * CLEAR. This command causes all FILEDEF definitions to be cleared, except for those defined with the PERM option.

If you want any FILEDEF definitions to remain in effect after a language processor has executed, define them with the PERM option.

- An ABEND occurs. When you enter your next command, ABEND recovery takes place. ABEND recovery clears all previously specified FILEDEF definitions, including those for which the PERM option was specified. All the FILEDEF definitions must be respecified after each ABEND recovery.

Examples

```
FILEDEF MACLIB DISK SYS1 MACLIB B1 (MEMBER ABEND)
```

When the CMS file SYS1 MACLIB or, if the B disk is an OS disk, the OS data set SYS1.MACLIB is opened, the read/write pointer is set to the start of the partitioned data set member ABEND.

```
FILEDEF ASSEMBLE DISK TEST ASSEMBLE B1 DSN SAMPLE OS DATA
```

When an OS macro or a CMS command references the fileid TEST ASSEMBLE B1, it gets information from the OS data set SAMPLE.OS.DATA. For example, the command ASSEMBLE TEST assembles the OS data set SAMPLE.OS.DATA.

```
FILEDEF SYSPRINT PRINTER (PERM RECFM F BLOCK 132)
```

Whenever SYSPRINT is referred to in your program, the output written is spooled to the virtual printer. The file is fixed-format with a logical block length of 132. The file definition is not removed by the command FILEDEF * CLEAR.

```
FILEDEF PRINTOUT TERMINAL (UPCASE NOCHANGE
```

If a file definition for the ddname PRINTOUT does not exist, one is established. Output written to PRINTOUT is displayed at the terminal in uppercase.

```
FILEDEF DISK DISK NAME OLDFILE
```

All I/O for a file with a ddname of DISK is directed to a disk file with a file identifier of NAME OLDFILE.

```
FILEDEF DISKFILE DUMMY
```

When any I/O command is issued to a disk file with a ddname of DISKFILE, the real I/O operation is not performed.

```
FILEDEF NEWMAST TAP2 (9TRACK DEN 1600 RECFM FB LRECL 050 BLOCK 3000
```

I/O commands issued to a file with ddname of NEWMAST are directed to the tape located at logical unit TAP2. The tape is 9-track, and recording is done at 1600 bpi. The tape is in fixed-block format with a logical record length of 50 and a physical blocksize of 3000.

```
FILEDEF CMSLIB DISK OS1 MACLIB * DSN OS DATA 1 (LRECL 80 BLOCK 800 RECFM FB CONCAT)
FILEDEF CMSLIB DISK OS2 MACLIB * DSN OS DATA 2 (CONCAT)
FILEDEF CMSLIB DISK SYS1 MACLIB * (CONCAT)
GLOBAL MACLIB OS1 OS2 SYS1 MYLIB
```

This example shows how you can use the CONCAT option to cause CMS to search the OS maclibs OS.DATA.1, OS.DATA.2 and SYS1.MACLIB and the CMS maclib MYLIB to resolve maclib references in assembling the CMS file SAMPLE ASSEMBLE.

```
ACCESS 193 B
ACCESS 194 C
FILEDEF CMSLIB DISK ASP1 MACLIB * DSN ASP1 MACROS RL1 (RECFM FIXED BLOCK 3360 LRECL 80 CONCAT)
FILEDEF CMSLIB DISK ASP2 MACLIB * DSN ASP2 MACROS RL2 (CONCAT)
FILEDEF CMSLIB DISK SYS1 MACLIB * (CONCAT)
FILEDEF ASSEMBLE TEST SAMPLE B1 DSN TEST OS SAMPLE1
GLOBAL MACLIB ASP1 ASP2 SYS1 CMSLIB
ASSEMBLE TEST
```

This example shows how you can use FILEDEF to override the default FILEDEF commands CMS gives you for an assembly. The example also shows how to point to an OS disk containing OS source for TEST.OS.SAMPLE1 and the OS macro libraries ASP.MACROS.RL1, ASP.MACROS.RL2, and SYS1.MACLIB. The GLOBAL command accesses four macro libraries and the order of search for the macros needed during assembly, that is, ASP1 first, ASP2 next, and so on. The first three libraries are OS libraries and CMSLIB is a CMS macro library.

| Note: A disk does not have to be accessed at the time a FILEDEF is
| issued; however, if this is the case, a warning message is issued to the
| user informing him that the disk is not accessed.

Responses

```
ddname1   device1   [filename1  filetype1]
 .          .          .          .
 .          .          .          .
 .          .          .          .
ddnameN   deviceN   [filenameN  filetypeN]
```

A list of current definitions is displayed if the FILEDEF command is entered with no operands.

DMSFLD220R ENTER DATA SET NAME:

A FILEDEF command with the DSN ? operand was entered. Enter an OS data set name the form qual1.qual2....qualn; where qual1, qual2 through qualn are the qualifiers of an OS data set name.

DMSFLD704I INVALID CLEAR REQUEST

A CLEAR request was entered for a file definition that does not exist; no action is taken.

Other Messages and Return Codes

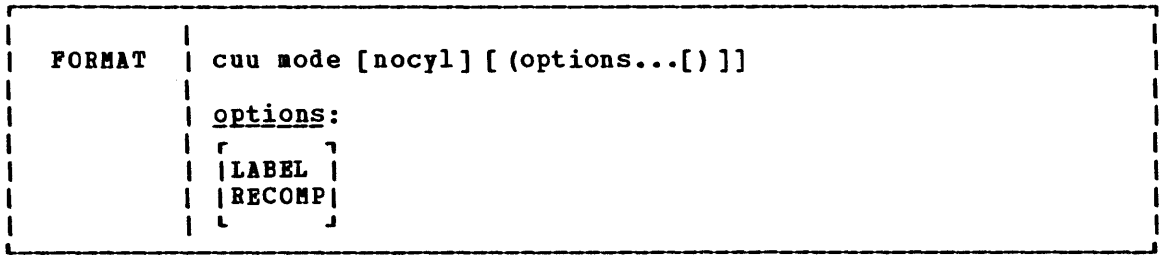
```
DMSFLD003E INVALID OPTION 'option'  RC=24
DMSFLD023E NO FILETYPE SPECIFIED  RC=24
DMSFLD027E INVALID DEVICE 'device name'  RC=24
DMSFLD029E INVALID PARAMETER 'param' IN THE OPTION 'option' FIELD RC=24
DMSFLD035E INVALID TAPE MODE  RC=24
DMSFLD050E PARAMETER MISSING AFTER DDNAME  RC=24
DMSFLD065E 'option' OPTION SPECIFIED TWICE  RC=24
DMSFLD066E 'option' AND 'option' ARE CONFLICTING OPTIONS  RC=24
DMSFLD070E INVALID PARAMETER 'param'  RC=24
DMSFLD221E INVALID DATA SET NAME 'data set name'  RC=24
DMSFLD224E FILEID ALREADY IN USE  RC=24
```

FORMAT

Use the FORMAT command to:

- Initialize a virtual disk area in the CMS format.
- Count the number of cylinders on a virtual disk.
- Write a label on a virtual disk.
- Reset the number of cylinders on the virtual disk.

This command can be used with a virtual 3340, 3330, 2314, or 2319 direct access storage device. The format of the FORMAT command is:



where:

cuu is the virtual device address of the virtual disk to be formatted.

Note that 000 is not a valid address.

mode is the filemode letter to be assigned to the specified device address. Valid filemode letters are A, B, C, D, E, F, G, Y, and Z. This field must be specified.

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.

Options

LABEL writes a label on the disk without formatting the disk. A six-character label is written on cylinder 0, track 0, record 3 of the virtual disk. A prompting message requests a six-character disk label (less than six is left-justified, blank padded).

RECOMP changes the number of cylinders on the disk which are available to the user to the actual number of minidisk cylinders or to the number specified by nocyl, whichever is less. If nocyl is not specified, all cylinders are used.

Note: If neither RECOMP nor LABEL is specified, 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.

Examples:

```
FORMAT 191 A  25
```

Initializes 25 cylinders of the disk located at virtual address 191 in CMS format.

```
FORMAT 192 B  25 (RECOMP)
```

Changes the number of cylinders available at virtual address 192 to 25 cylinders.

```
FORMAT 193 C (LABEL)
```

Writes a label on the disk at virtual address 193. Respond to the prompting message with a six-character label.

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: any existing files will be 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- to six-character label.

DMSFOR705I DISK REMAINS UNCHANGED

The response to message, DMSFOR603R, was 'NO'.

DMSFOR732I 'nnn' CYL FORMATTED ON DISK 'mode(ccu)'

A formatting operation has been done on nnn cylinders of the disk at virtual address ccu.

DMSFOR733I FORMATTING DISK 'mode'

The disk represented by mode letter 'mode' is being formatted.

DISK 'mode (cuu)': 'n' FILES, 'n' BLOCKS, 'n' LEFT (OF 'n'), nn% FULL ('n' CYL)

This message gives the extent and other information about a disk when a RECOMP operation has been done.

Other Messages and Return Codes

```
DMSFOR003E INVALID OPTION 'option'  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 'param'  RC=24
DMSFOR113S cuu NOT ATTACHED  RC=100
DMSFOR114S 'cuu' IS AN UNSUPPORTED DEVICE TYPE  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
```

GENDIRT

Use the GENDIRT command to create a CMS auxiliary directory. The auxiliary directory contains the name and location of modules which would otherwise significantly increase the size of the resident directory, thus increasing search time and storage requirements. By using GENDIRT to create 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. 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.

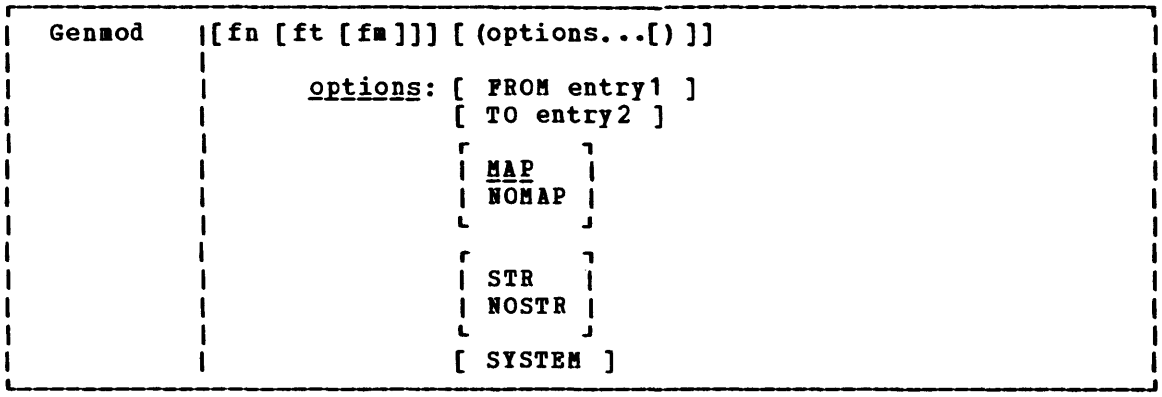
Note: See the VM/370: System Programmer's Guide for information on creating auxiliary directories and for further requirements for using the targetmode option.

Error Messages and Return Codes

```
DMSGND002W FILE 'fn ft fm' NOT FOUND  RC=4
DMSGND021E ENTRY POINT 'name' NOT FOUND  RC=28
DMSGND022E NO DIRECTORY NAME SPECIFIED  RC=24
DMSGND070E INVALID PARAMETER 'param'  RC=24
```

GENMOD

Use the GENMOD command to generate absolute core-image files. The format of the GENMOD command is:



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.

ft is the filetype of the MODULE file being created. If specified, ft must be MODULE.

fm is the filemode of the MODULE file being created. If fm is not specified, the file is written on the primary read/write disk.

Options

If conflicting options are specified, the last one entered is used.

FROM entry1 specifies an entry point or a control section name, which is the starting virtual storage location from which the core-image copy is generated.

If FROM is not specified, the starting virtual storage location of the module is either the address of fn (if it is an external name), or the address of the first external name encountered during the loading process. This is not necessarily the lowest address loaded. If you have any external references before your START or CSECT instructions, you must specify FROM entry1 to load your program properly.

TO entry2 specifies an entry point or a control section name, which is the ending virtual storage location from which the core-image copy is generated.

MAP includes a load map in the MODULE file.

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 LCADMOD 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 is subsequently loaded (see the CMS LOADMOD command). 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 of available user storage.

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 is to be loaded at a location other than the default load address.

SYSTEM indicates that when the MODULE is subsequently loaded, it is to have a storage protect key of zero.

Notes:

1. Before the file is written, undefined symbols are set to location 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.

2. If you load a program into the transient area you should issue the GENMOD command with the STR option. Be careful if the program issues OS GETMAIN or FREEMAIN because your program, plus the amount of storage obtained via GETMAIN, cannot exceed two pages (8192K). It is recommended that you do not issue a GETMAIN from a transient area.

Error Messages and Return Codes

```
DMSMOD001E NO FILENAME SPECIFIED  RC=24
DMSMOD002E FILE 'fn ft' NOT FOUND  RC=28
DMSMOD003E INVALID OPTION 'option'  RC=24
DMSMOD005E NO (FROM TO) ENTRY SPECIFIED  RC=24
DMSMCD021E 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 '...'  RC=24
DMSMOD084E INVALID USE OF 'FROM' AND 'TO' OPTIONS  RC=24
DMSMOD105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSMOD109S VIRTUAL STORAGE CAPACITY EXCEEDED  RC=104
```

GLOBAL

Use the GLOBAL command to specify which CMS libraries are to be searched when processing subsequent CMS commands. The GLOBAL command remains in effect for an entire CMS session unless it is explicitly canceled or until another GLOBAL command is entered. There are no default libraries, so the command must be libraries if any libraries are to be used. The GLOBAL command verifies the existence of the libraries and issues a warning message if a specified library does not exist. The format of the GLOBAL command is:

```
┌──────────────────────────────────────────────────────────────┐
│ GLobal  │ { MACLIB } [libname1 ... libname8]                 │
│         │ { TXTLIB }                                         │
└──────────────────────────────────────────────────────────────┘
```

where:

MACLIB — allows the specification of the macro libraries that are to be used 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, FILEDEF must be issued for the data set before you issue GLOBAL. (See the description of the FILEDEF CONCAT option for more information.)

TXTLIB — allows 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 OS SVC 8 is issued).

Note: 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.

libname1... — is the filename of up to eight libraries. If the MACLIB form of the GLOBAL command is used, the filetypes of all files specified must be MACLIB. If the TXTLIB form of the command is used, the filetypes of all files specified must be TXTLIB. 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.

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 must redefine it (again, via FILEDEF), since the language processors clear your previous definition of the library.

Another means for reusing a library is the PERM option of the FILEDEF command. If you use PERM, you must clear that FILEDEF before you issue another GLOBAL for a different library.

Examples

```
GLOBAL MACLIB ACCESS SYSMAC
```

The system searches the ACCESS MACLIB and SYSMAC MACLIB files for missing macros during compilations.

```
GLOBAL TXTLIB CONVERT FLOAT
```

The system searches the CONVERT TXTLIB and FLOAT TXTLIB files for missing subroutines during subsequent LOAD and INCLUDE operations.

```
GLOBAL MACLIB
```

Cancels the effect of any previous GLOBAL MACLIB libname command.

Error 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
```

INCLUDE

Use the INCLUDE command to read one or more TEXT files (containing relocatable object code) from disk and to load them into storage, establishing the proper linkages between the files. INCLUDE is normally used to resolve references left unresolved by a previous LOAD or INCLUDE command. Refer to Figure 21 for a description of the handling of unresolved references. A LOAD command must have been previously issued for the INCLUDE command to produce desirable results. The format of the INCLUDE command is:

```
 INclude   fn... [(options...[)]]
           options:  [CLEAR  ]  [       {entry}]  [ORIGIN hexloc]
                     [NOCLEAR]  [RESET   {  *  }]
                                [       {     }]

                [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 language processor commands.

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, the starting point is reset to the location of the first file or to the address specified as an operand of an END card, LDT card, or ENTRY card.

ORIGIN hexloc — begins loading the program at the location specified by hexloc. The variable, hexloc, is a hexadecimal number of up to eight 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.

MAP — adds information to the LOAD MAP file.

NOMAP — does not add any information to the LOAD MAP file.

TYPE — displays the load map of the files at the terminal, as well as writing it on the primary disk. This option is valid only if MAP is specified or implied.

NOTYPE — does not display the LOAD MAP file at the terminal.

INV — prints invalid card images in the LOAD MAP file.

NOINV — does not print invalid card images in the LOAD MAP file.

REP — prints replace 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 printing of replace 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 the examples.

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.

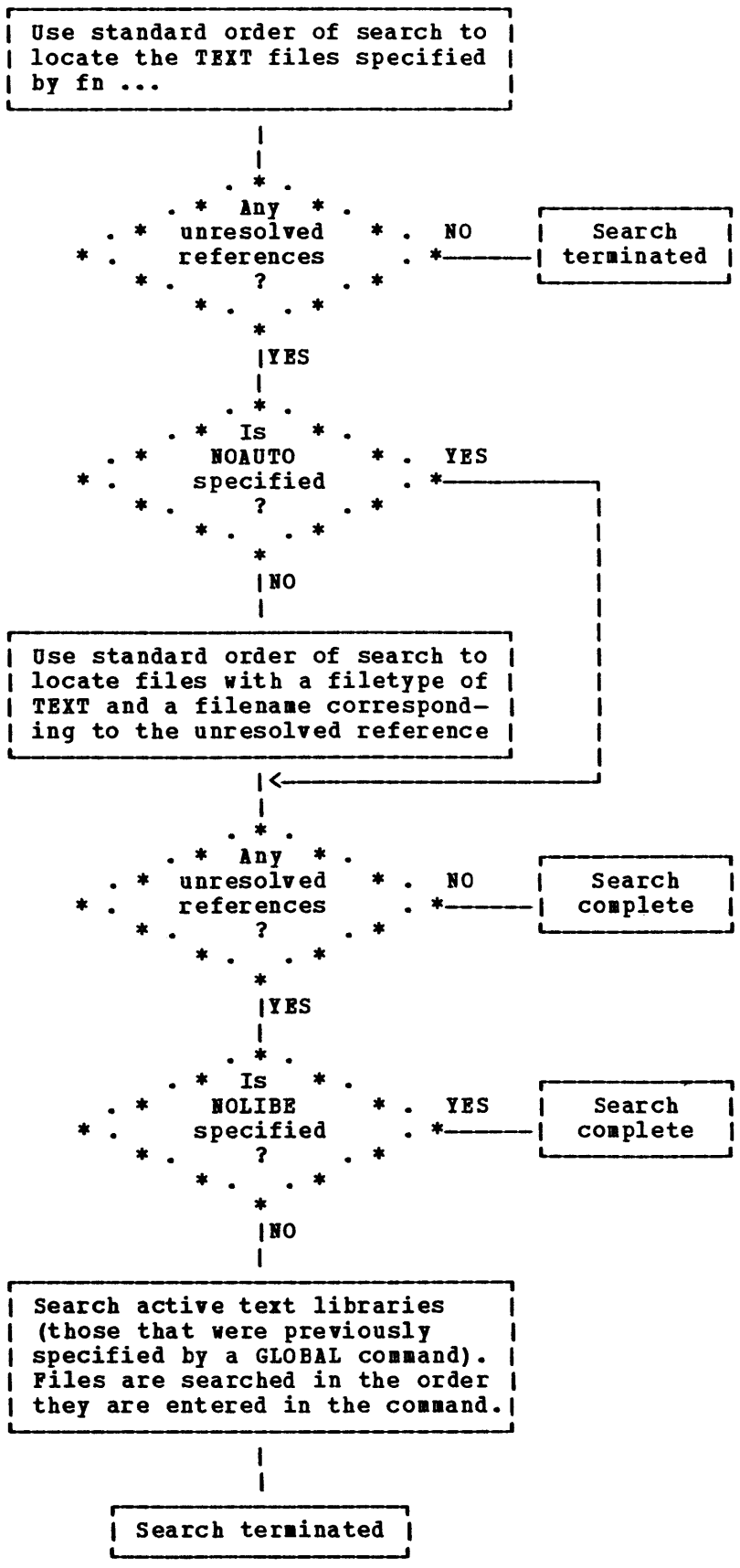


Figure 21. Resolution of Unresolved References

Examples

```
INCLUDE MAIN SUBI DATA (RESET MAIN MAP START)
```

Brings the files named MAIN TEXT, SUBI TEXT, and DATA TEXT into real storage and appends them to files which 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, AUTO. During execution of the INCLUDE command, the file named MYSUB TEXT is appended to MYPROG TEXT. The following options are in effect:

```
MAP, NOLIBE, NOREP, NOTYPE, INV, AUTO
```

Note: After you IPL CMS, at least one LOAD command must be issued before INCLUDE can be used with predictable results.

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 file LOAD MAP. The invalid card is ignored and loading continues.

CONTROL CARD - ...

A loader or library-search control statement was encountered (that is, ENTRY or LIBRARY). See the description of the LOAD command for the use of ENTRY and LIBRARY control cards. This response is placed in the LOAD MAP file only.

Other Messages and Return Codes

```
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 'param' IN THE OPTION 'option' FIELD  RC=24
DMSLIO055E NO ENTRY POINT DEFINED  RC=40
DMSLIO056E FILE 'fn ft' CONTAINS INVALID RECORD FORMATS  RC=32
```

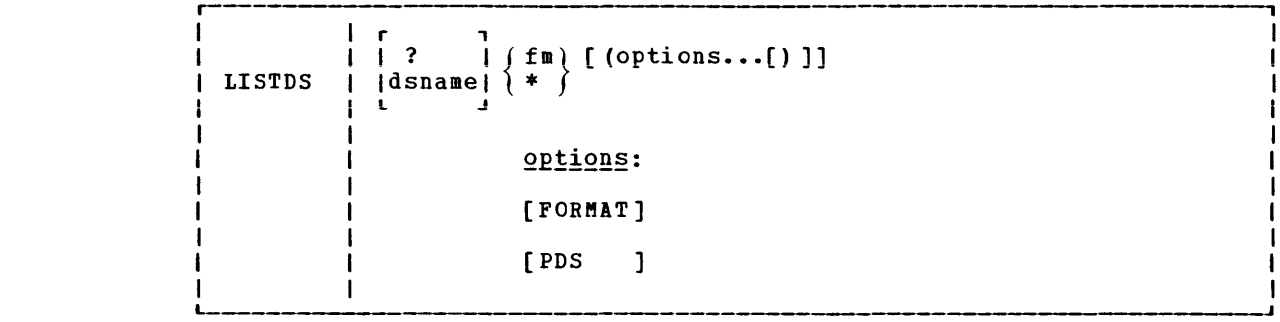
INCLUDE

```
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
```

LISTDS

Use the LISTDS command to list at your terminal information describing the data sets residing on an OS disk, or the files on a DOS disk.

The format of the LISTDS command is:



where:

? indicates that you want to enter the OS data set name or DOS file-id from your terminal. If you enter the question mark (?), CMS requests that you enter the OS data set name or DOS file-id exactly as it appears in the data set or file. Do not 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.qualn]

   where qual1, qual2, through qualn are qualifiers for the data set name or file-id. Using this form, you can specify data set names or file-ids that contain embedded special characters such as blanks and hyphens.

dsname is the OS data set name or DOS file-id and takes the form:

   qual1[qual2 qualn]

   where qual1, qual2, through qualn are the qualifiers for the data set name or file-id. Each qualifier must not exceed 8 characters and must be separated from other qualifiers by blanks, not periods. (This form can be used for DOS file-ids only if they comply with the OS naming convention of one- to eight-byte qualifiers separated by periods, to a maximum of 44 characters, including periods.) For example, for an OS data set or DOS file named MY.FILE.IN, you enter:

```
LISTDS MY FILE IN
```

fm is the filemode of the data set or file being listed.

* indicates that you want the data set name (or file-id) and filemode listed for all of the data sets (or files) on all of the OS or DOS disks currently accessed.

Options

FORMAT
FO — displays the date, disk label, filemode, and data set name for the OS data set as well as the RECFM, LRECL, BLKSIZE, and DSORG operands. For a DOS file, LISTDS displays the date, disk label, filemode, and file-id, but gives no information about the RECFM, LRECL, and BLKSIZE operands (two blanks appear for each); DSORG is always PS.

PDS — displays the member names of references to OS Partitioned Access Method (PAM) data sets.

Example

The following is an example of LISTDS with the FORMAT and PDS options and the output resulting from the command.

```
     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
```

Response

DMSLDS220R ENTER DATA SET NAME:

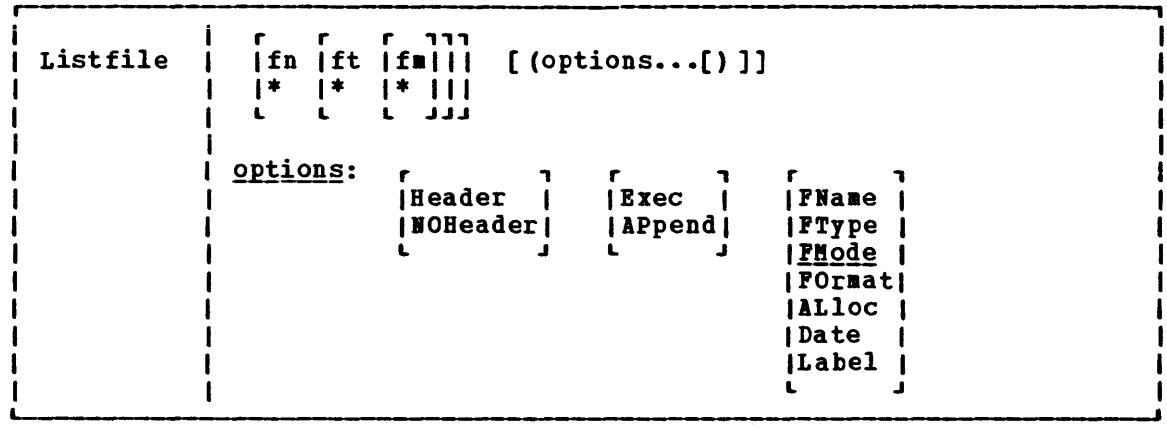
The LISTDS command with the ? operand was issued. Enter an OS data set name in the form qual1.qual2.qualn; where qual1, qual2, through qualn are the qualifiers of the OS data set name.

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
DMSLDS221E INVALID DATA SET NAME  RC=24
DMSLDS222E I/O ERROR READING 'data set name' FROM OS DISK  RC=28
DMSLDS223E NO FILENAME SPECIFIED  RC=24
DMSLDS231E I/O ERROR READING VTOC FROM ...-DISK (OS)  RC=28
```

LISTFILE

Use the LISTFILE command to obtain specified information about your CMS files residing on accessed disks. The information may be either displayed at the terminal or used to create a special EXEC file on disk. All operands are optional; if no operands are specified, a list of default information about each file on your primary read/write disk is displayed at the terminal. The format of the LISTFILE command is:



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.

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.

fm is the filemode of the files for which information is to be collected. If this field is omitted, only the primary disk is searched. If an asterisk is coded, all disks are searched.

Note: An asterisk (*), immediately preceded by any number of characters for fn or ft, searches for the specified characters as the leading characters for that identifier. For example, LISTFILE ABC* ASSEMBLE prints the identifiers for all ASSEMBLE files with filenames beginning with ABC.

Output Format Options

HEADER includes column headings in the listing. HEADER is the default if any of the "Supplemental Information" options are specified. The format of the heading is:

```
FILENAME FILETYPE FM FORMAT 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 file of 80-character records (one record for each of the files which satisfies the given file identifier) on the primary disk. The file that contains these records is called CMS EXEC A1. If a file with this name already exists, the existing one is erased and a new one is created (unless the APPEND option is specified, in which case the existing file is retained and the new entries are appended to it). The EXEC procedure thus created contains two symbolic variables, &1 and &2. This CMS EXEC file is used with the EXEC command, but it can also be processed as any other file (that is, printed, displayed, edited, added to, changed, and so forth). The header is not included in the file.

APPEND appends the EXEC list created to the existing CMS EXEC A1 file. If the EXEC option is specified instead of APPEND, any existing CMS EXEC file is erased and replaced by the file created by this LISTFILE command. If this option is specified and no CMS EXEC file exists, one is created.

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
```

Option priority is 2.

LABEL includes the label of the disk on which the file resides in the list. Option priority is 1.

Example

```
LISTFILE * ASSEMBLE * (LABEL)
```

LABEL is the lowest priority option; therefore, all other options are also in effect. All information about the files with a filetype of ASSEMBLE is displayed at the terminal. The header is displayed because LABEL is a supplemental information option.

The following is displayed at the terminal:

```
FILENAME FILETYPE  FM  FORMAT    RECS BLOCKS   DATE    TIME   LABEL
                       ┌─┐
                       |F|
fn       ASSEMBLE  fm  |V| lrecl norecs noblks mm/dd/yy hh:mm volid
.          .       .   └─┘   .     .      .       .       .     .
.          .       .   .     .     .      .       .       .     .
.          .       .   .     .     .      .       .       .     .
```

One entry is displayed for each file with a filetype of ASSEMBLE.

where:

fn — is the filename of the file.

ASSEMBLE — is the filetype specified in the command.

fm — is the filemode of the file.

┌─┐
|F| — is the file format: F = fixed length
|V| V = 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 created.

hh:mm — is the time (hours:minutes) that the file was created.

volid — is the volume serial number of the virtual disk on which the file resides.

Responses

If the EXEC option is not specified, the requested information is displayed at the terminal.

Error 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 'param'  RC=24
DMSLST105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
```

Note: 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.

LOAD

Use the LOAD 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.

Notes:
1. The LOAD command requires a read/write A-disk to contain a work file.
2. If you are loading a program into the transient area see Note 2 for the GENMOD command.

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 language processors.

Note: If you have a program that issues a dynamic load command (OS SVC 8) for a subroutine that (1) is a CSECT in a CMS text library member and (2) contains VCONs for entry points not in the same member, that subroutine must be explicitly loaded with the program that uses it. Otherwise, the entry point returned by the CMS LOAD command is unpredictable.

Options

If conflicting options are specified, the last one entered is in effect.

CLEAR clears the load area in storage to binary zeroes before the object files are loaded.

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 either (1) the entry point of the first file loaded, (2) the address on an END card, (3) the location of a name occurring on an LDT card, or (4) the location of the operand on an ENTRY statement. If * is entered the results are the same as if the RESET option were omitted.

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 eight characters. If TRANS is specified, the file is loaded into the CMS nucleus transient storage locations. If this option is not specified, loading begins at the first available user storage location.

Note: Any program loaded into the transient area must have a starting address of X'E000'.

MAP creates the file LOAD MAP on the primary disk.

NOMAP does not create the file LOAD MAP.

TYPE displays the LOAD MAP file at the terminal. This option is valid only if the MAP option is in effect.

NOTYPE does not display the LOAD MAP file at the terminal.

INV includes the invalid card images in the LOAD MAP file.

NCINV does not include the invalid card images in the LOAD MAP file.

REP includes the replace statements in the LOAD MAP file.

NCREP does not include the replace statements in the LOAD MAP file.

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 GLOBAL command.

NCLIBE 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, START can be specified. LOAD then transfers control to an entry point in the program. The default entry point is (1) the address specified in the operand field of the first END statement containing a non-blank operand field, (2) the address of a name on an LDT statement, (3) the beginning of the first file loaded (if all END statements in the TEXT files contain blank operand fields), or (4) the ENTRY specified.

DUP displays warning messages at your terminal when a duplicate CSECT is encountered during processing. The duplicate CSECT is not loaded.

NODUP does not display warning messages at your terminal when duplicate CSECTs are encountered during processing. The duplicate CSECT is not loaded.

FILES CREATED BY THE LOAD COMMAND: The LOAD command produces one temporary workfile:

DMSLDR SYSUT1

This temporary work file is placed on the read/write A-disk, which must be available.

LOAD MAP FILE: Unless the NOMAP option is specified, a load map is created on the primary disk each time the LOAD command is issued. A load map is a file that contains the location of control sections and entry points of files loaded into storage. It may also contain messages and card images for invalid cards or replace cards that exist in the loaded files. This load map is normally created as a file with the file identification LOAD MAP. Only one such file may exist on the primary disk. 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 INVALID CARD in the LOAD MAP file. To suppress this listing in the load map, the NOINV option must be specified.

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, the NOREP option must be specified.

DUPLICATE CSECTS: 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.

LOADER CONTROL STATEMENTS: Five types of control statements can be added to a TEXT file. These are the set location counter (SLC), the include control section (ICS), the replace (REP), the ENTRY, and the LIBRARY statements. These are used to set the virtual storage location where the LOAD command begins placing the file, to make corrections and additions to the relocatable object code in virtual storage once the file is loaded, to specify entry points, and to specify references that are not to be resolved. These statements can be added to the TEXT files already punched and can then be read back in, or they can be added using the EDIT command.

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 22. 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.

   If there are blanks in both columns 7-12 and 17-22, or the symbolic name has not yet been defined, the response INVALID CARD xxx...xxx is displayed or, depending on the option (NOINV or INV) specified, is written in the LOAD MAP file. If only the symbolic address is to be used, columns 7-12 must be left blank or be all zeros. If only the absolute address is to be used, columns 17-22 must be left blank.

Column	Contents
1	Load control statement identification (12-2-9 punch). Identifies this as a statement acceptable to the loader.
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 22. SLC 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 23.

Column	Contents
1	Load control statement identication (12-2-9 punch). Identifies this as a statement acceptable to the loader.
2-4	LDT -- identifies type of statement.
5-16	Not used.
17-24	Blank or entry name (left justified padded with blanks to 8 characters).
25-80	Not used.

Figure 23. 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 24. An ICS statement must be placed at the front of the file or TEXT file.

Column	Contents
1	Load control statement identification (12-2-9 punch). Identifies this as a statement acceptable to the loader.
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 these 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 24. ICS 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 25. 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	Load control statement identication (12-2-9 punch). Identifies this as a statement acceptable to the loader.
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 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 25. REP Statement Format

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 26. 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 26. ENTRY Statement Format

The loader selects the entry point for the loaded program according to the following hierarchy:

1. From the parameter list on the START command.
2. From the last RESET operand in the LOAD or INCLUDE command entered.
3. From the last ENTRY statement in the input.
4. From the last LDT statement in the input.
5. From the first assembler- or compiler-produced END statement that specifies an entry point if no ENTRY statement is in the input.
6. 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. For example:

   ```
   ENTRY GO
   ```

   where GO is defined as the external name of the first instruction to be executed when the program is loaded. The address of the instruction, indicated by the symbolic name GO, is specified by the loader as the starting point of the program when it is executed.

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 27. 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.

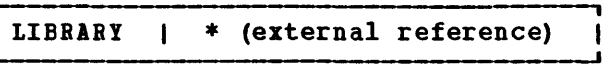


Figure 27. LIBRARY Statement Format

Example

```
LIBRARY * (SINE)
```

The * specifies the never-call function. SINE is an external reference in the output. As a result, if SINE is unresolved after input processing, no automatic library call is made.

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.

CONTROL CARD - ...

A loader or library-search control statement (that is, ENTRY or LIBRARY) was encountered. This response is placed in the LOAD MAP file.

Other Messages and Return Codes

```
DMSLIO001E NO FILENAME SPECIFIED  RC=24
DMSLIO002E FILE 'fn ft' NOT FOUND  RC=28
DMSLIO003E INVALID OPTION 'option'  RC=24
DMSLIO004E ENTRY POINT 'name' NOT FOUND  RC=40
DMSLIO005E NO option SPECIFIED  RC=24
DMSLIO021E ENTRY POINT 'name' NOT FOUND  RC=40
DMSLIO029E INVALID PARAMETER 'param' IN THE OPTION 'option' FIELD  RC=24
DMSLIO055E NO ENTRY POINT DEFINED  RC=40
DMSLIO056E FILE 'fn ft' CONTAINS INVALID RECORD FORMATS  RC=32
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
```

LOADMOD

Use the LOADMOD command to bring a disk file into storage. The file must be in absolute core-image format as created by the GENMOD command. The format of the LOADMOD command is:

```
┌──────────────────────────────────────────────────────────────┐
│ LOADMod │ fn [ft fm]                                         │
└──────────────────────────────────────────────────────────────┘
```

where:

fn is the filename of the file to be loaded into storage.

ft is the filetype of the file to be loaded. If supplied, the filetype must be MODULE.

fm is the filemode of the module to be loaded.

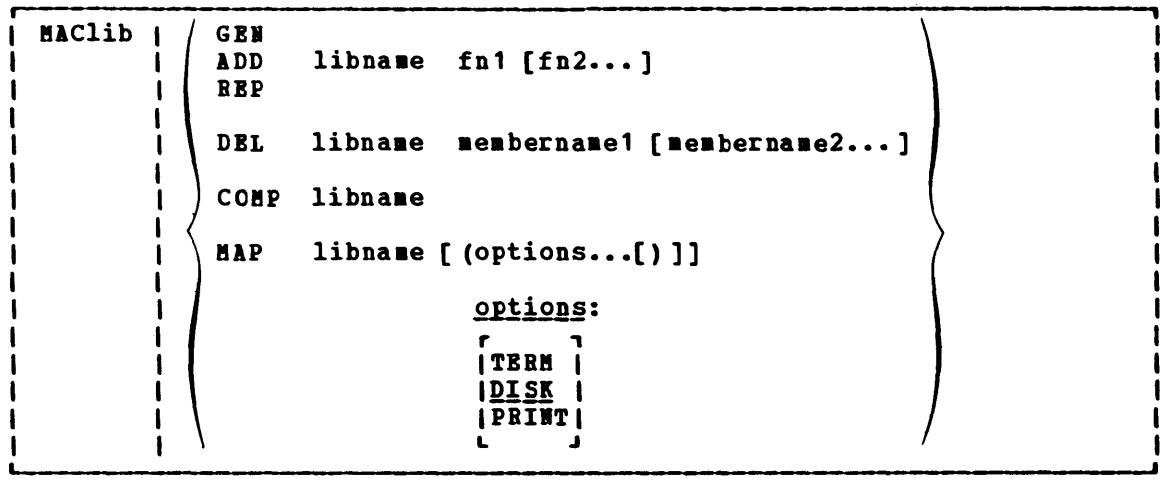
If filetype and filemode are not supplied, the standard order of search is used to locate a file with the specified filename and a filetype of MODULE.

Error Messages and Return Codes

```
DMSMOD001E NO FILENAME SPECIFIED  RC=24
DMSMOD002E FILE 'fn ft' NOT FOUND  RC=28
DMSMOD032E INVALID FILETYPE 'ft'  RC=24
DMSMOD037E DISK 'mode' IS READ/ONLY  RC=36
DMSMOD040E NO FILES LOADED  RC=40
DMSMOD070E INVALID PARAMETER 'param'  RC=24
DMSMCD104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSMOD109S VIRTUAL STORAGE CAPACITY EXCEEDED  RC=104
DMSMOD116S LOADER TABLE OVERFLOW  RC=104
```

MACLIB

Use the MACLIB command to create and modify macro libraries. A macro library consists of members (macro definitions) and a dictionary which contains the name of the macro (member name), its size, and its location relative to the beginning of the library. The MACLIB command checks only the macro definitions of MACRO and MEND statements for errors. The format of the MACLIB command is:



where:

GEN — generates a CMS macro library.

ADD — adds members to an existing macro library.

REP — replaces existing members in a macro library.

DEL — deletes members from a macro library.

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.

In a MACRO file, the macro name is taken from a prototype statement within the macro.

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 fn. 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.

membername1... is the name of a macro which exists in a macro library.

MAP Options

TERM
DISK
PRINT — is the device to contain the output generated by the MAP function. Valid device names are DISK (disk file), PRINT (virtual spooled printer), and TERM (terminal). If no device is specified, DISK is assumed. Since these are the only options allowed in the MACLIB command, only the first word after the left parenthesis is examined. If the DISK option is specified, the information is written to a file named 'libname MAP A1'. If a file with that name previously existed it is replaced by the new file.

The following paragraphs describe each of the functions of the MACLIB command.

GEN Function

The GEN (generate) function creates a new CMS macro library with the filename and filetype you specify on the MACLIB command. If a macro library with the same filename already exists, it is erased and replaced by the new macro library. The new file is created from input files specified by fn1 fn2 ... For example:

```
MACLIB  GEN  OSMAC  ACCESS  TIME  PUT  REGEQU
```

Creates a new macro library with the file identification OSMAC MACLIB from macros existing in the files with the file identifiers:

ACCESS {MACRO / COPY}, TIME {MACRO / COPY}, PUT {MACRO / COPY} and REGEQU {MACRO / COPY}

If a file named OSMAC MACLIB already exists, that file is erased.

Assume that the files ACCESS MACRO, TIME COPY, PUT MACRO, and REGEQU COPY exist and contain macros in the following form.

ACCESS MACRO	TIME COPY	PUT MACRO	REGEQU COPY
GET	*COPY TTIMER TTIMER	PUT	XREG
PUT	*COPY STIMER STIMER		YREG

The resulting file OSMAC MACLIB contains the following members:

```
GET
PUT
TTIMER
STIMER
PUT
REGEQU
```

Note: The PUT macro, which appears twice in the input to the command, also appears twice in the output. The MACLIB command does not check for duplicate macro names. If, at a later time, the PUT macro is requested from OSMAC MACLIB, the first PUT macro encountered in the dictionary is used.

ADD Function

The ADD function appends the members described by the macro definition files (fn1 fn2...) to an existing macro library.

```
MACLIB  ADD  OSMAC  DCB
```

Assume that OSMAC MACLIB was created by the example in the explanation of the GEN function and the file DCB COPY exists as follows.

```
*COPY DCB
 DCB macro definition
*COPY DCBD
 DCBD macro definition
```

The resulting OSMAC MACLIB contains the following members:

```
GET
PUT
TTIMER
STIMER
PUT
REGEQU
DCB
DCBD
```

REP Function

REP (replace) is effectively a delete function followed by an ADD function. REP deletes the dictionary entry for the macro definition in the files specified by fn1 fn2 ... It then appends the new macro definitions to the macro library and creates new dictionary entries. For example, assume that a macro library MYMAC MACLIB contains the members A, B, and C, and that the following command is entered:

```
MACLIB  REP  MYMAC  A  C
```

The files represented by file identifiers A MACRO and C MACRO each have one macro definition. After execution of the command, MYMAC MACLIB contains members with the same names as before, but the contents of A and C are different.

DEL Function

The DEL (delete) function removes the specified macro name from the macro library dictionary and compresses the dictionary so there are no unused entries. The macro definition still occupies space in the library, but since no dictionary entry exists it cannot be accessed or retrieved. If you attempt to delete a macro for which two macro definitions exist in the macro library, only the first one encountered is deleted. For example:

```
MACLIB  DEL  OSMAC  GET  PUT  TTIMER  DCB
```

deletes macro names GET, PUT, TTIMER, and DCB from the dictionary of the macro library named OSMAC MACLIB. Assume that OSMAC exists as in the ADD function example. After the above command, OSMAC contains the following members:

```
STIMER
PUT
REGEQU
DCBD
```

COMP Function

Execution of a MACLIB command with the DEL or REP functions can result in unused space within a macro library. The COMP (compress) function is used to compress a macro library (that is, remove any macros for which there is no dictionary entry). This function uses a temporary data set named MACLIB CMSUT1. For example, the command:

```
MACLIB  COMP  MYMAC
```

Compresses the library MYMAC MACLIB.

MAP Function

The MAP function creates a list containing the name of each macro in the dictionary, the size of the macro, and its position within the macro library. You can specify the device to which the list is to be written. Acceptable devices are:

DISK — the list is placed in a file with the file identification 'libname MAP A1'.

PRINT — the list is spooled to the printer.

TERM — the list is displayed at the terminal.

Response

When the MAP operand is specified on the MACLIB command, the response is:

```
membername size location
```

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
DMSLBM070E INVALID PARAMETER 'param'  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  MEMBER   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. Two types of modules contain no load map and therefore produce an error message when they are specified in a MODMAP command. They are (1) CMS transient area modules and (2) MODULE files created with the GENMOD command using the NOMAP option. 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.

Responses

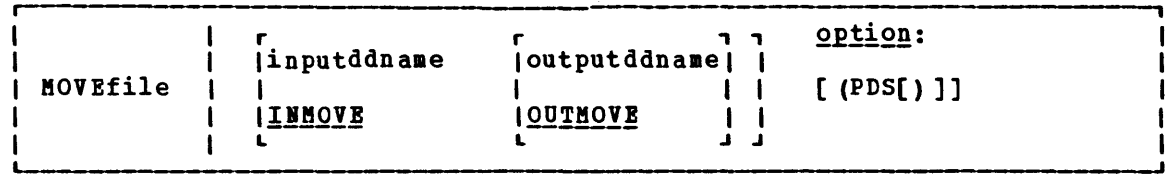
The load map associated with the file is displayed at the terminal.

Other 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 'param'  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 command accepts two ddnames as arguments. The FILEDEF command must have specified devices or disk files for these ddnames. The command moves data records from the device, or file, specified by the first ddname to the device, or file, specified by the second ddname. The format of the MOVEFILE command is:



where:

inputddname is the ddname representing the input file definition. If ddname is not specified, the default input filename, INMOVE, is used.

outputddname is the ddname representing the output file definition. If ddname is not specified, the default output filename, OUTMOVE, is used.

Option:

PDS moves all of the members of the CMS MACLIB or TXTLIB or of an OS partitioned data set into separate CMS files, each with a filename equal to the member name and a filetype equal to the filetype of the output file definition (FILEDEF).

Note: Normally, the FILEDEF command is used to establish device characteristics for the ddname specified with the command. If the FILEDEF command was not issued, the OS macro simulation routines supply default characteristics. For example, if the ddname is X a default FILEDEF X DISK FILE X A1 command is executed.

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 the record format specification, that specification is taken from the default list shown in Figure 28. 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: if the record format is F or U, the logical record length equals the blocksize; if the record format is V, the logical record length equals the blocksize minus 4.

Device	Input ddname		Output ddname	
	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 28. Default Device Attributes for MOVEFILE Command

Example

The existing file whose file definition ddname is NEWMAST is moved to the file whose file definition ddname is OLDMAST.

```
MOVEFILE NEWMAST OLDMAST
```

The following sequence of CMS commands move an OS STOW macro file from an OS partitioned data set SYS1.MACLIB or a CMS file SYS1 MACLIB to the CMS file STOW MACRO.

```
ACCESS 195 B/A
FILEDEF TEST1 DISK SYS1 MACLIB B1 (MEMBER STOW)
FILEDEF MACRO DISK STOW MACRO
MOVEFILE TEST1 MACRO
```

The following sequence of CMS commands moves all the members of an OS partitioned data set SYS1.MACLIB or a CMS file SYS1 MACLIB into separate CMS files each with a filename equal to its member name and a filetype of MACRO.

```
ACCESS 195 B
FILEDEF TEST2 DISK SYS1 MACLIB B1
FILEDEF MACRO DISK
MOVEFILE TEST2 MACRO (PDS)
```

For more examples of how to move OS data sets and DOS files to CMS files using MOVEFILE, see "Using OS Programs and Macros under CMS" in Section 4.

Responses

DMSMVE225I PDS MEMBER 'membername' MOVED

The specified member of an OS partitioned data set was moved successfully to a CMS file.

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 command refers to a terminal. This message requests the input data; a null line terminates input.

DMSMVE708I DISK FILE 'FILE ddname A1' ASSUMED FOR DDNAME 'ddname'

No FILEDEF command was issued for a ddname specified on the MOVEFILE command. As a result the MOVEFILE issues a FILEDEF for that ddname.

If the input ddname is undefined, the file must exist. The block size and record format are taken from this file.

If the output ddname is undefined, the disk file is created on the A-disk with the same characteristics as those of the input file.

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
DMSMVE048E INVALID MODE 'mode'  RC=24
DMSMVE070E INVALID PARAMETER 'param'  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
```

PRINT

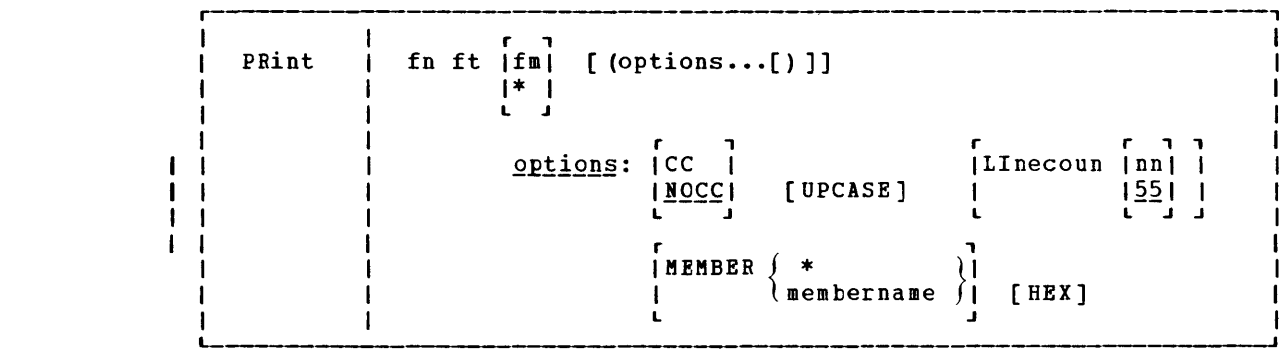
Use the PRINT command to print a CMS file on the spooled virtual 1403 or 3211 printer. The file may contain carriage control characters and may have either fixed- or variable-length records, but no record may exceed 132 characters for a 1403 or 150 characters for a 3211. There are two exceptions to this:

1. 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.

2. 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.

Note: An option may not be in effect, even though it is specified. See the discussion of the PRINT options for information on overriding options.

The format of the PRINT command is:



where:

fn is the filename of the file to be printed. This field must be specified.

ft is the filetype of the file to be printed. This field must be specified.

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 encountered, with the given filename and filetype, is printed. If fm is not specified, the primary 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 or 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 performs the necessary carriage control to eject a new page and print 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 {name}	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 * 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 not in effect, even if specified, and even if the filetype is LISTING.
LINECOUN [nn] [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 and has a default value of 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.

Examples

```
PRINT  MYLIB  MACLIB  (MEMBER GET)
```

Spools the contents of the member GET in file MYLIB MACLIB to the printer. The first character of each record in the file is not used for carriage control.

```
PRINT  OLDMAST  NAME  (CC)
```

Spools the contents of the file OLDMAST NAME to the printer and uses the first character of each record in the file for carriage control.

Responses

None.

The 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.

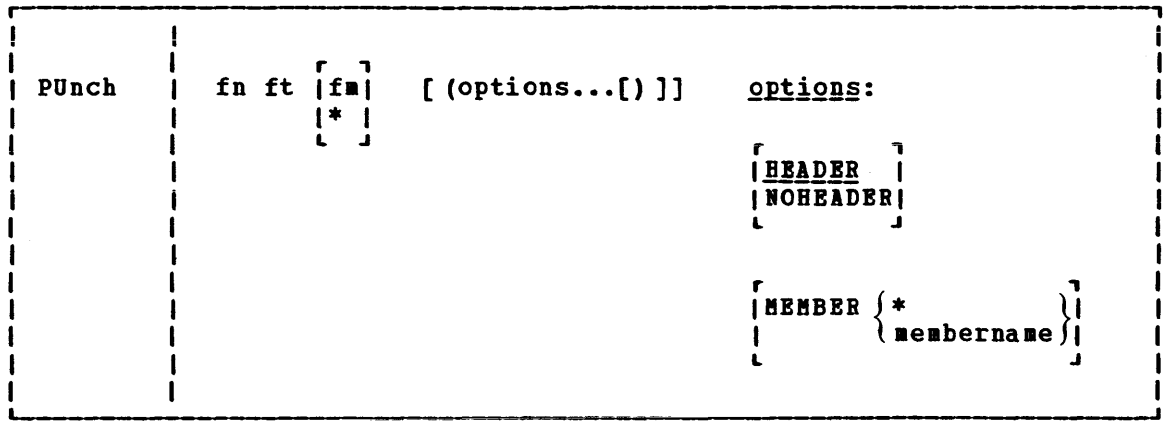
Other Messages and Return Codes

```
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 'param' 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 'fm'  RC=24
DMSPRT054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSPRT062E INVALID * IN FILEID  RC=20
DMSPRT070E INVALID PARAMETER 'parm'  RC=24
DMSPRT104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSPRT123S ERROR PRINTING FILE 'fn ft fm'  RC= 100
```

PUNCH

Use the PUNCH command to spool a specified CMS disk file to the punch. PUNCH accepts fixed- or variable-length records as long as no record exceeds 80 characters. Records with less than 80 characters are padded with blanks on the right. Records longer than 80 characters are rejected. The format of the PUNCH command is:



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 file must be specified.

fm is the filemode of the file to be punched. If this field is specified as an asterisk (*), the standard order of search is followed and the first acceptable file encountered is punched. If fm is not specified, the primary disk and its extensions are searched.

Options

HEADER
H — inserts a control card in the punched output preceding the specified file. This control card identifies the file for a subsequent READCARD command to restore the file to a disk. If the filetype is TXTLIB and the MEMBER membername option is specified, the header card contains TEXT as the filetype. If the filetype is MACLIB and the MEMBER membername option is specified, the header contains MEMBER as the filetype. A macro library member originates from a file with a filetype of either MACRO or COPY. The control card format is shown in Figure 29.

NOHEADER
NOH — does not insert a header control card in the punched deck.

MEMBER {* membername}
MEM {* membername}
punches library members. If the filetype is MACLIB or TXTLIB, this option are specified. If * is entered, all

individual members of that macro or text library are punched. If membername is specified, only that member is punched.

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 29. Header Card Format

Example

```
PUNCH  NEWMAST  TRANS  (NOH)
```

Spools the file NEWMAST TRANS to the punch. No header card is punched preceding the output deck.

Responses

None.

If the command completes without error (the file was successfully spooled), the READY message appears. The file is now under control of CP spooling functions.

Other Messages and Return Codes

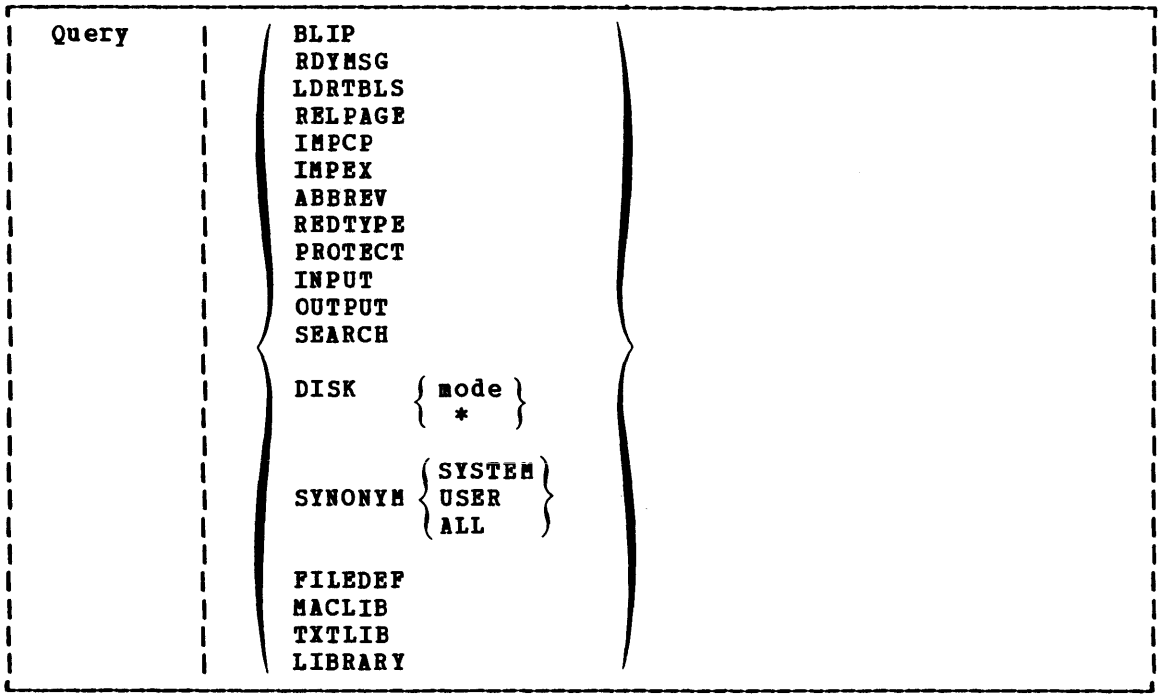
```
DMSPUN002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSPUN003E INVALID OPTION 'option'  RC=24
DMSPUN008E DEVICE 'cuu' {INVALID OR NONEXISTENT|UNSUPPORTED DEVICE TYPE}
           RC=36
DMSPUN013E MEMBER 'name' NOT FOUND IN LIBRARY  RC=32
DMSPUN033E FILE 'fn ft fm' IS NOT A LIBRARY  RC=32
DMSPUN039E NO ENTRIES 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 certain information about the CMS virtual machine environment. Information which may be obtained includes:

- The state of any of the virtual machine characteristics which are controlled by the CMS SET command.
- File definitions (set with the FILEDEF command) which are in effect.
- The status of the disks attached to your virtual machine.

The format of the QUERY command is:



Options for SET Command Functions:

BLIP displays the BLIP character(s).

Response: BLIP = { xxxxxxxx | OFF }

RDYMSG displays the RDYMSG format.

Response: RDYMSG = { LMSG | SMSG }

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 = {ON | OFF}

where:

ON releases pages.
OFF retains pages.

IMPCP displays the status of implied CP command indicator.

Response: IMPCP = {ON | OFF}

where:

ON passes commands not recognized by CMS to CP.
OFF flags commands not recognized by CMS.

IMPEX displays status of implied EXEC indicator.

Response: IMPEX = {ON | OFF}

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 = {ON | OFF}

where:

ON accepts minimum truncations for CMS commands.
OFF does not accept minimum truncations.

REDTYPE displays the status of the REDTYPE indicator.

Response: REDTYPE = {ON | OFF}

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 documented 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 = {ON / OFF}

where:

ON means CMS nucleus protection is in effect.
OFF means CMS nucleus protection is not in effect.

INPUT displays the contents of your input translate table if one is specified.

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 your output translate table if you have specified one.

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

Options for CMS Disk Status Functions:

SEARCH displays the search order of all CMS disks currently accessible.

Response: volid vaddr mode {R/O / R/W}
. . .
. . .
. . . .
. . . .
. . . .

DISK mode displays the status of the single disk represented by 'mode'.

Response: mode (vaddr): nn FILES, nnnn REC IN USE, nnnn LEFT (OF nnnn), nn% FULL (n CYL), type {R/O / R/W}

If the disk is an OS-formatted disk, the response is:

mode(vaddr): (n CYL), R/O-OS

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: 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 Command	Shortest 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 Command	User Synonym	Shortest 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: Same as SYNONYM SYSTEM and SYNONYM USER.

FILEDEF displays all file definitions in effect.

Response: ddname device [fn [ft]]

If there are no user file definitions in effect, the following message is displayed at the terminal:

```
NO USER DEFINED FILEDEF'S IN EFFECT
```

MACLIB displays the names of all files with a filetype of MACLIB which are to be searched for macro definitions.

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 which are to be searched for unresolved references.

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 name of all macro and text library (files with filetypes MACLIB and TXTLIB) which are to be searched.

Response: MACLIB = { libname... | NONE }

TXTLIB = { libname... | NONE }

Error Messages and Return Codes

```
DMSQRY005E NO 'option' SPECIFIED  RC=24
DMSQRY014E INVALID FUNCTION 'function'  RC=24
DMSQRY026E INVALID PARAMETER 'param' FOR 'function' FUNCTION  RC=24
DMSQRY047E NO FUNCTION SPECIFIED  RC=24
DMSQRY070E INVALID PARAMETER 'param'  RC=24
```

READCARD

Use the READCARD command to read data records from the virtual card reader (spool input device) and to create files on disk containing the data records. The data records must be fixed length and normally contain 80 characters, although they may contain up to 151 characters. Records less than 80 characters long cannot be read. If a file exists on disk with the same identifiers as the one to be created, it is erased.

Any number of files may be entered through the card reader, each immediately preceded by a READ control card specifying the filename, filetype, and optionally the filemode. The READ control card is shown in Figure 30. The header card supplied when HEADER is specified with the CMS PUNCH command is a valid READ control card.

All files that are logically grouped together must have the same record length. The READ control cards are displayed at the terminal as they are encountered, and interpreted just as if filename and filetype had been entered from the terminal. Each READ control card ends the preceding file, and the reader end-of-file indication ends the last file.

Your files must be spooled to the virtual reader before a READCARD command can be issued. You can send your real card deck to your virtual card reader using the CP ID card in front of your real card deck. This card takes the form:

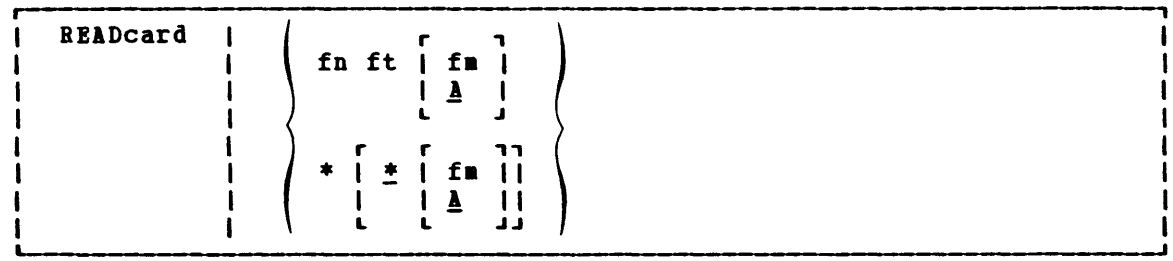
```
| {ID     } userid [CLASS c] NAME {fn ft  }
| {USERID }      [CLASS A]      {dsname}
|
```

where:

ID USERID	is a keyword that must begin in column one.
userid	is the user's identification (userid), limited to eight characters.
CLASS c	is the optional class field. If CLASS is not specified, class A is the default.
NAME {fn ft dsname}	is the name field. If the first form is specified, the filename and filetype are each limited to eight characters. The filename and filetype must be separated by a blank. If the second form is used, the dsname field is limited to 24 characters.

Note: Only the CLASS n operand is optional. All fields must be separated by at least one blank. The keyword ID or USERID must start in column one.

You need not be logged on at the time the decks are transferred to your virtual reader. If more than one file is spooled to the virtual reader, more than one READCARD command must be issued to process all the logical files. If SPOOL RDR CONT is issued, all the logical files in the virtual reader are treated as a single file. Again, the files that are treated as one must all have the same record length. The format of the READCARD command is:



where:

fn is the filename of the file to be read. An asterisk may be coded in this field.

ft is the filetype of the file to be read. An asterisk may be coded in this field if fn was coded as an asterisk.

fm is the filemode of the file to be read. If this field is omitted or specified as an asterisk (*), A 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.

Forms of the READCARD Operands

```
READCARD MYFILE ASSEMBLE
```

If filename and filetype are specified with the READCARD command, only one file is read. The READ control cards are ignored.

```
READCARD *
```

If the file identification is to be entered in the card stream, a single asterisk must be specified with the READCARD command. If this form of the command is specified, and the first card in the input stream is not a valid READ control card, a file named READCARD CMSUT1 A1 is set up to contain all data read until a valid READ control card is encountered. If fn ft fm was specified, the file is written on the specified disk with a mode number matching the filemode number from the control card. If the filemode number is specified, that number is used. If filemode is omitted, A is assumed.

```
READCARD * *
```

Two asterisks accept the filename, filetype, and filemode number from the READ control card, but use a filemode letter of A.

```
READCARD * * B1
```

Two asterisks with a mode specified accept the filename and filetype identifiers from the READ control card, but use the filemode specified.

Whatever is specified on the READCARD command line is used. Whatever is not specified is taken from the READ control card, except the filemode letter, which defaults to A. If there is no READ control card, a READ control card

```
:READ READCARD CMSUT1 A1
```

is assumed.

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	fn	Filename of the file.
16	1	blank	
17-24	8	ft	Filetype of the file.
25	1	blank	
26-27	2	fm	Filemode of the file.
28-80	53	anything	

Figure 30. Format of the READ Control Card

Responses

After the command READCARD * is issued, control cards encountered in the input card stream are displayed at the terminal.

DMSRDC701I NULL FILE

The spooled card reader contains no records after the control card.

DMSRDC702I READ CONTROL CARD IS MISSING. FOLLOWING ASSUMED: 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.

When a READCARD control card is encountered, the first 72 columns of a card are displayed at the terminal.

Other Messages and Return Codes

DMSRDC008E DEVICE 'cuu' {INVALID OR NONEXISTENT|UNSUPPORTED DEVICE TYPE} RC=36
DMSRDC042E NO FILEID SPECIFIED RC=24
DMSRDC054E INCORRECT 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 active disk when it is no longer needed. An ACCESS command must have been previously entered for the specified disk. The format of the RELEASE command is:

where:

cuu is the virtual device address of the disk that is to be released.

Note that 000 is not a valid address.

mode is the mode of the disk to be released.

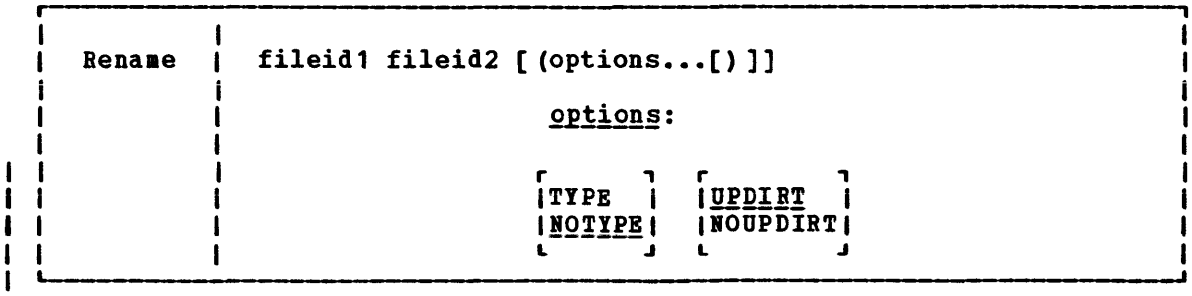
Note: If a disk is accessed more than once, the RELEASE cuu command releases all instances of cuu. The system disk cannot be released.

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 'param'  RC=24
```

RENAME

Use the RENAME command to change the file identification of one or more files. The RENAME command may not be used for a file which is located on a read-only disk. The format of the RENAME command is:



where:

fileid1 is the file identification of the original file whose name is to be changed. All components of the file (filename, filetype, and filemode) must be coded, either with a name or an asterisk. If an asterisk is coded in any field, any file which satisfies the other qualifications is renamed. For example, if fileid1 is coded as A * A1, all files on the A disk with a filename of A are renamed.

fileid2 is the new file identification 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.

Options

TYPE
T — displays at the terminal the new identifiers of all files renamed. The file identifiers are displayed only when an * is specified for one or more of the file identifiers (fn, ft or fm) in fileid1.

NOTYPE
NOT — suppresses displaying at the terminal 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. Normally, Master File Directories are updated at the completion of each CMS command that affects disk files.

Responses

newfn newft newfm

The new filename, filetype, and filemode of each file altered is displayed only 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 original file.

If fileid2 is the name of an existing file, an error message is generated.

Other 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
DMSRNM051E INVALID MODE CHANGE  RC=24
DMSRNM054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSRNM062E INVALID * IN OUTPUT FILEID  RC=20
```

| 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 DMSRNM002E FILE fn ft fm NOT FOUND error message is not
| issued.

RUN

Use the RUN command to initiate an automated series of functions on a file. The RUN command can compile, load, and start execution of the specified file, depending upon the filetype. The acceptable filetypes are: EXEC, MODULE, TEXT, and those required by the language processors. The RUN command is an EXEC procedure: if it is executed from within an EXEC file, it must be preceded by the EXEC command name. The format of the RUN command is:

```
| RUN | fn [ft [fm]] [(args...)]                                    |
```

where:

fn is the filename of the file to be manipulated. This field must be entered.

ft is the filetype of the file to be manipulated. If filetype is not 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 order). 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. Then, LOAD and START are called to initiate program execution. The filetype must be specified if filemode is specified. The valid filetypes and resulting action for 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 begins execution of the program at the entry point equal to fn.
TEXT	The LOAD command brings the file into 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, TESTFORT, or FORTHX, whichever is found first.
FREEFORT	The GOFORT module is called to process the file.
COBOL	The COBOL processor module that is called is COBOL or TESTCOB, whichever is found first.
PLI PLIOPT	The PLIOPT processor module is called to process the file.

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 one or more user arguments to be used during execution. If compiling or loadig is to be performed, it is assumed that the default options for those functions are in effect. You can specify up to 13 arguments in the RUN command, provided they fit on a single input line. These arguments are used during the operation of an EXEC file, or during execution of a MODULE or TEXT file. The arguments are set up as a string of doublewords, one argument per doubleword. The address of this string is passed to the specified file at execution time. Each argument is left-justified, and any argument more than eight characters long is truncated on the right. With an EXEC file, any arguments specified in the RUN command replace the corresponding &n operands in the individual commands of the EXEC file.

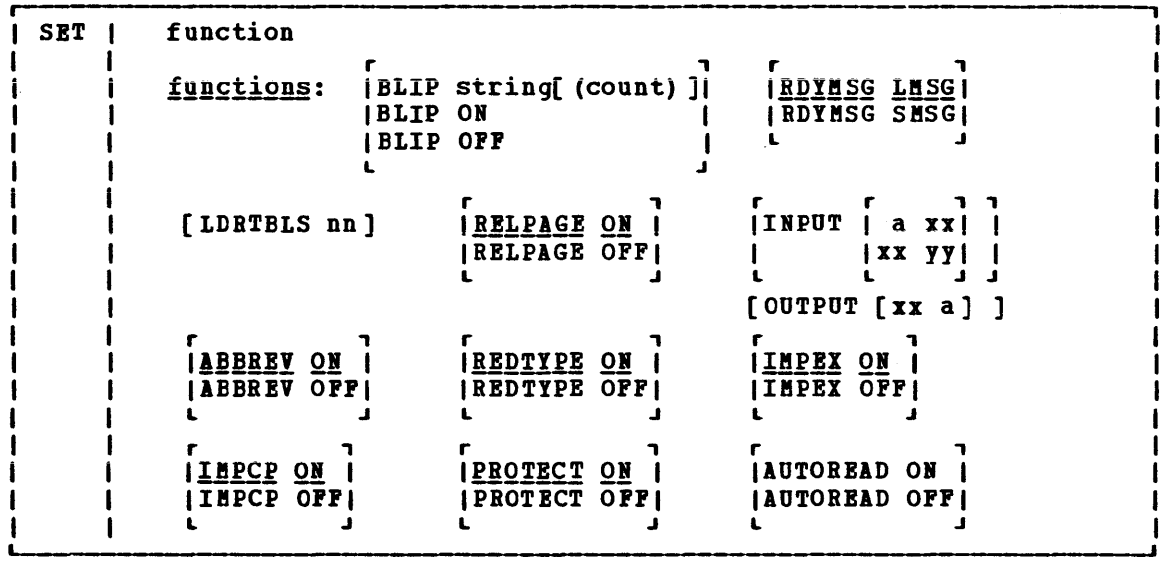
With a file whose filetype is other than EXEC, the arguments are placed in a string as described above. The address of the string is passed to the specified file at execution time. The end of the argument list is denoted by a X'FF' in the first byte of the argument field.

Error Messages and Return Codes

```
DMSRUN001E NO FILENAME SPECIFIED  RC=24
DMSRUN002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSRUN032E INVALID FILETYPE 'ft'  RC=24
DMSRUN048E INVALID MODE 'fm'  RC=24
DMSRUN070E INVALID PARAMETER 'param'  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:



where:

Functions

BLIP string[(count)] defines the characters which are displayed at the terminal to indicate every two CPU seconds of real (or virtual) execution time. Up to eight characters can be defined, and if trailing blanks are desired, the count field must be used. 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 for the 3767 terminal. Thus, another character must be defined as the BLIP character for the 3767 if you want the BLIP function.

BLIP OFF turns off BLIP. OFF is the default for graphics devices.

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 CPU time, Real CPU 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. This number can be changed with the SET LDRTBLS nn command provided that: (1) nn is a decimal number less than 128, (2) the virtual machine has enough storage available to allow nn pages to be used for loader tables, and (3) the system has not started using storage just below the LDRTBLS. If these three conditions are met, nn pages are set aside for loader tables. If you plan to change the number of pages allocated for loader tables, you should do so as soon after IPL as possible.
RELPAGE ON	releases page frames of storage and sets them to binary zeros 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 previous paragraph 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	allows 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, the SET OUTPUT command must be issued 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.
ABBREV ON	allows the system abbreviation or your own abbreviation (if one is available) to invoke a system command. The SYNONYM command makes the system and user abbreviations available.
ABBREV OFF	invokes a command only when the full system command name or the full user synonym (if one is available) is entered. 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 documented 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. To execute an EXEC file, the EXEC command name must be issued.
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 recognized by CMS.
PROTECT ON	protects the CMS nucleus against writing within its storage area.
PROTECT OFF	does not 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 non-display, non-buffered terminals.
AUTOREAD OFF	specifies that you do not want a console READ until you depress 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 a user disconnects from one type of terminal and reconnects on to another type, the AUTOREAD status will remain unchanged.

Error Messages and Return Codes

```
DMSSET014E INVALID FUNCTION 'function'  RC=24
DMSSET026E INVALID PARAMETER 'param' FOR 'function' FUNCTION  RC=24
DMSSET031E LOADER TABLES CANNOT BE MODIFIED  RC=40
DMSSET047E NO FUNCTION SPECIFIED  RC=24
DMSSET061E NO TRANSLATION CHARACTER SPECIFIED  RC=24
DMSSET070E INVALID PARAMETER 'param'  RC=24
```

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 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. The format of the SORT command is:

```
| SORT | fileid1 fileid2 |
```

where:

fileid1 is the file identification (filename, filetype, filemode) of the file containing the records to be sorted.

fileid2 is the file identification (filename, filetype, filemode) of the new output file to contain the sorted records.

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. Pass one creates an ordered pointer table in virtual storage. Pass two 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:

$$NR = \frac{VMSIZE - 132K}{14 + NC}$$

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 file NAME ADDRESS 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.

$$NR = \frac{VMSIZE-132K}{14 + NC} = \frac{320K-132K}{14 + 14} = \frac{188K}{28} = \frac{192,512}{28} = 6875$$

Responses

DMSSRT604R ENTER SORT FIELDS:

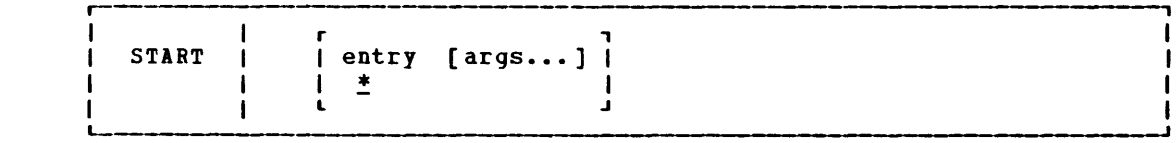
You are requested to enter SORT control fields. You should enter them in the form described in "Entering Sort Control Fields."

Other Messages and Return Codes

```
DMSSRT002E FILE 'fn 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 DEFINED  RC=24
DMSSRT054E INCOMPLETE FILEID SPECIFIED  RC=24
DMSSRT062E INVALID * IN FILEID  RC=20
DMSSRT063E NO LIST ENTERED  RC=40
DMSSRT070E INVALID PARAMETER 'param'  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
```

START

Use the START command to begin execution of programs that were previously loaded, and to pass the address of a string of arguments to that program. The format of the START command is:



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. The default entry point is either the address specified in the operand field of the first END control statement containing a non-blank operand field, or the beginning of the first file loaded if all END control statements in the TEXT files contains blank operand fields. The default entry point can be changed by specifying the RESET option on the INCLUDE command, when loading additional files.

args... — are arguments to be passed to the started program. If user arguments are specified, entry or * 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.

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

DMSLIO074I EXECUTION BEGINS...

is displayed when the designated entry point is validated.

Other Messages and Return Codes

DMSLIO021E ENTRY POINT 'name' NOT FOUND RC=40
DMSLIO055E NO ENTRY POINT DEFINED RC=40

STATE

Use the STATE command to verify the existence of a CMS file. The format of the STATE command is:

```
| STATE | fn ft [fm]
```

where:

fn is the filename of the file whose existence is to be verified. This field must be specified.

ft is the filetype of the file whose existence is to be verified. This field must be specified.

fm is the filemode of the file whose existence is to be verified. If this field is omitted, all your disks are searched.

Note: If * is specified for fn, ft, and/or fm the first file found satisfying the rest of the fileid is used.

If the filemode refers to an OS or DOS disk, the STATE command assumes that the filename and filetype are related to an OS data set name or DOS file-id through a previous FILEDEF. If an associated FILEDEF command was not issued, you receive a FILE NOT FOUND message.

Responses

DMSSTT227I PROCESSING VOLUME 'no' IN DATA SET 'data set name'

The specified data set has multiple volumes; the volume being processing 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.

Other 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 'param'  RC=24
DMSSTT229E UNSUPPORTED OS DATA SET, ERROR 'code'
```

Note: You can invoke the STATE command from the terminal, from an EXEC file, or as a function from a program. If STATE is invoked as a function or from an EXEC file that has the &CONTROL NOMSG option in effect, the DMSSTT002E FILE fn ft fm NOT FOUND error message is not issued.

SVCTRACE

Use the SVCTRACE command to trace and record information about supervisor calls occurring in your virtual machine.

The information recorded 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 and floating-point registers before branching to the SVC-called program and after returning from it, and 16 words of the parameter list which existed when the SVC was issued.

To terminate tracing previously established by the SVCTRACE command, issue the HO or SVCTRACE OFF commands. Both SVCTRACE OFF and HO cause all trace information recorded up to the point they are issued to be printed on the virtual spooled printer. On 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 command HX or log off the control program before termination of tracing set by SVCTRACE, the switches are cleared automatically and all recorded trace information is printed on the virtual spooled printer. The format of the SVCTRACE command is:

where:

ON starts tracing all SVC instructions issued within CMS.

OFF stops SVC tracing.

The printer trace output consists of the following:

- 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 with 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.

Responses

A variety of information is printed whenever the

```
SVCTRACE ON
```

command is issued.

The first line of trace output starts with a -, +, 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 * applies.

* indicates information recorded after processing a CMS SVC which 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 minus sign (-). If the first character of this line is a plus sign or asterisk (+ or *), PSW2 represents the PSW which 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 minus sign (-), 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 Register 15 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 only printed 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 31 summarizes the types of SVC trace output.

Identification	Comments
{+ / - / *} N/D	The SVC and the routine which 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 31. Summary of SVC Trace Output Lines

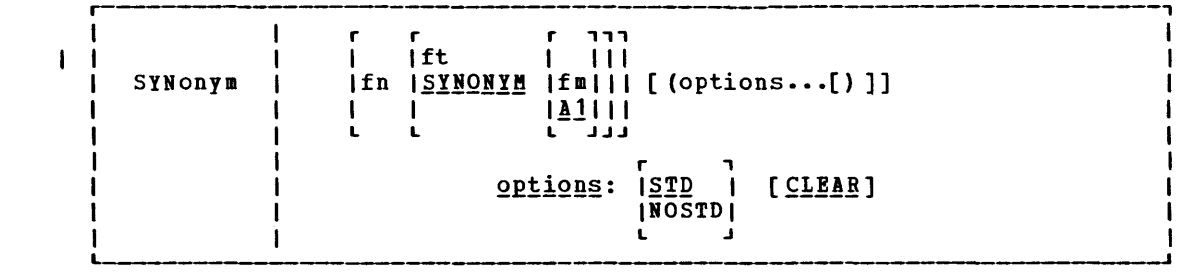
Error Messages and Return Codes

```
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, the CMS 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 operative. The format of the SYNONYM command is:



where:

fn is the filename of the file you created to contain the synonyms.

ft is the filetype of the file containing your synonyms. The filetype must be SYNONYM; of omitted, SYNONYM is assumed.

fm is the filemode of the file containing your synonyms. If omitted, A1 is assumed.

Options

STD specifies that standard CMS abbreviations are operative.

NOSTD standard CMS abbreviations are not to be used. (But 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.

The SYNONYM command specified with no operands can be used to nullify the synonyms invoked by a preceding SYNONYM command; that is, you can "turn off" a table of synonyms.

Note: 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 for which you are creating a synonym.

usersynonym is the synonym you are creating for a CMS command. 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. For more details on syntax rules for commands, see "Section 2: VM/370 CP and CMS Command Languages."

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 contains:

```
MOVEFILE  MVIT
```

The synonym MVIT can be entered as a command name to execute the MOVEFILE command. It cannot be truncated since no count is specified.

```
ACCESS GETDISK 3
```

The synonyms GET, GETD, GETDI, GETDIS, or GETDISK can be entered as the command name instead of ACCESS.

The Relationship Between the SET ABBREV and SYNONYM Commands

There is a system synonym abbreviation table for the FILEDEF command. The default values of the SET and SYNONYM commands are such that the system synonym 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.

Assume that the user-defined table has the file identification USERTAB SYNONYM A. Further assume that this synonym table has the following 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 32 lists the forms of the system command and user synonym available for the various combinations of the SET ABBREV and SYNONYM commands.

Responses

DMSSYN712I NO SYNONYMS (DMSINA NOT IN NUCLEUS)

The system routine which handles SYNONYM processing is not in the system.

Other Messages and Return Codes

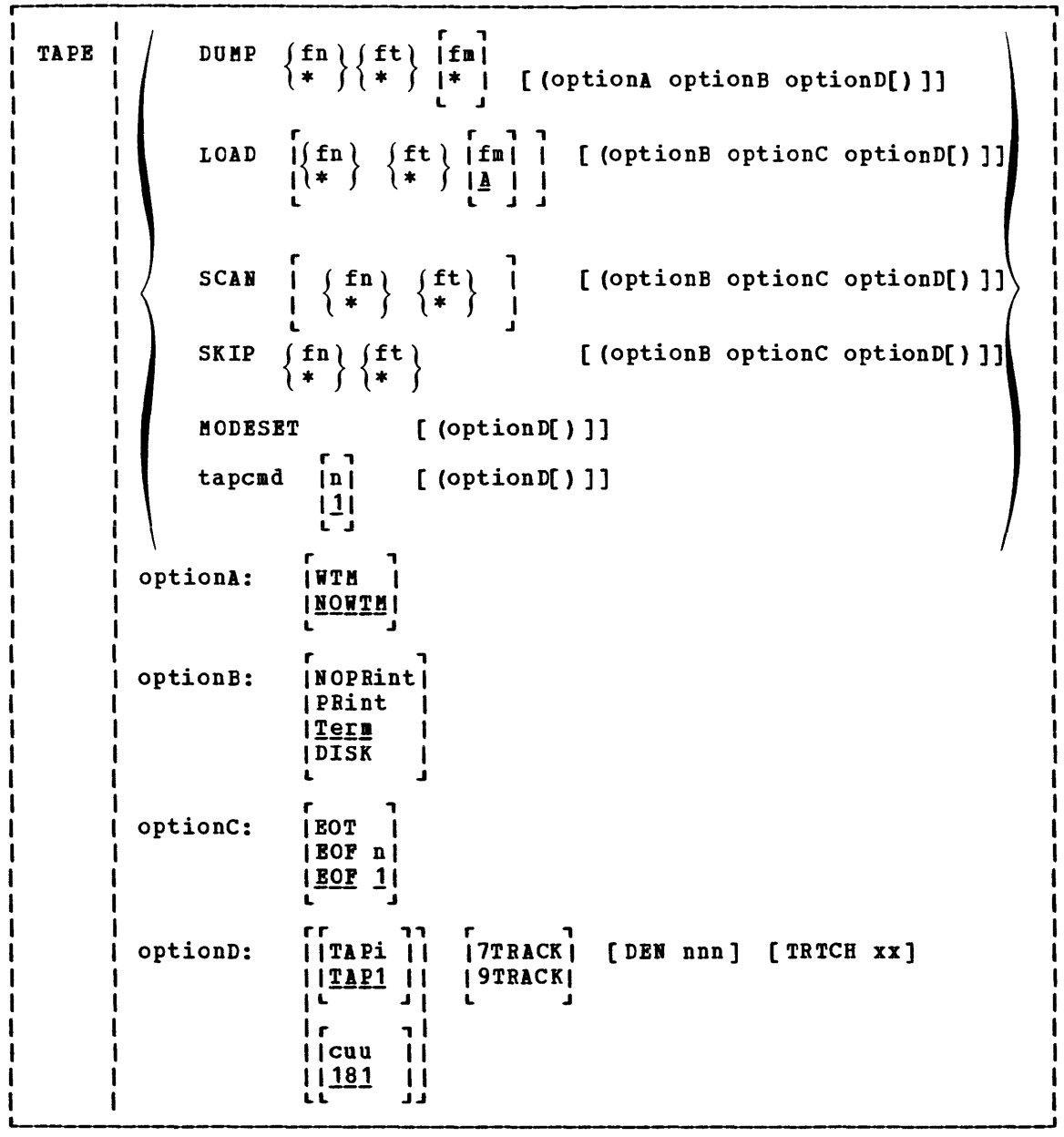
```
DMSSYN002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSSYN003E INVALID OPTION 'option'  RC=24
DMSSYN007E FILE 'fn ft fm' NOT FIXED, 80 CHAR RECORDS  RC=32
DMSSYN032E INVALID FILETYPE 'ft'  RC=24
DMSSYN056E FILE 'fn ft fm' CONTAINS INVALID RECORD FORMATS  RC=32
DMSSYN104S ERROR 'nn' READING FILE 'fn ft fm' 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 SET ABBREV ON SYN (CLEAR NOSTD SET ABBREV OFF SYN (CLEAR NOSTD	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.

Figure 32. System and User 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. TAPE is used solely for CMS files; therefore, the files on tape are 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:



where:

DUMP {fn|*} {ft|*} [fm|*]
: dumps one or more disk files to tape. The file identification must be specified. If the asterisk or mode

letter only is coded, all files that satisfy the resulting file identification are dumped.

The filename (fn) and filetype (ft) of the files to be dumped must be specified. The filemode (fm) of the files to be dumped is optional. The filemode letter indicates the source disk for dumping; the filemode number indicates that only files with that number are to be dumped. A blank filemode number indicates that all files satisfying the fn and ft specifications are to be dumped.

LOAD [{fn|*} {ft|*} [{fm|A}]]

writes tape files to disk. If file identification is specified, only that one file is loaded. The file identification is filename (fn), filetype (ft), and filemode (fm). If the option EOF n is specified and no file identification 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 resulting file identification 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. A blank filemode number indicates that all files satisfying the fn and ft specifications are to be loaded.

SCAN [{fn|*} {ft|*}]

displays at the terminal (unless NOPRINT, PRINT or DISK is specified) the names of the files on tape. If DISK is specified the list of file identifiers is written to a file named TAPE MAP. If file identification (filename, fn, and filetype, ft) is specified, scanning stops upon encountering that file. If not specified, scanning occurs over n tape marks as specified by the option EOF n.

SKIP {fn|*} {ft|*}

positions the tape at a specified point, depending upon other options and operands. If file identification (filename, fn, and filetype, ft) is entered, the tape is positioned after the specified file; if EOF n is entered, the tape is positioned after n tape marks.

MODESET sets the values specified by the DEN, TRACK, and TRTCH options. These values remain in effect for the specified tape 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):

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
RUN	rewind tape and unload
WTM	write n tape marks

Options

Note: If conflicting options are specified, the last one entered is in effect.

WTM — writes a tape mark on the tape after each file 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.

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.

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. Default is EOF 1.

TAPi
cuu — specifies the symbolic tape identification or the actual device address of the tape to be read from or written to. The default is TAP1 or 181. The unit specified by cuu must previously have been attached to your CMS virtual machine before any tape I/O operation can be attempted. Only symbol 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 nnnn — is the tape density where nnnn 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 dual-density drives, 1600 is the default.

TRTCH xx — is the tape recording technique for 7 track tape. If TRTCH is specified, 7TRACK is assumed. One of the following must be specified as xx:

xx	Meaning
0	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

Format of Tape Created by TAPE DUMP Command

Tape records written by the CMS TAPE DUMP command are 805 bytes long. The first character is a binary 2 (X'02'), followed by the characters CMS and an EBCDIC blank (X'40'), followed by 800 bytes of file data packed without regard for logical record length. In the final record, the character N replaces the blank after CMS, and the data area contains CMS file directory information.

TAPE Command Restrictions

If a tape contains large files that would not fit on disk, the tape load operation is terminated. To prevent this, when you dump the files, separate logical files by tape marks, then forward space to the appropriate file.

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.

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. The TAPE command creates a work file 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.

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
```

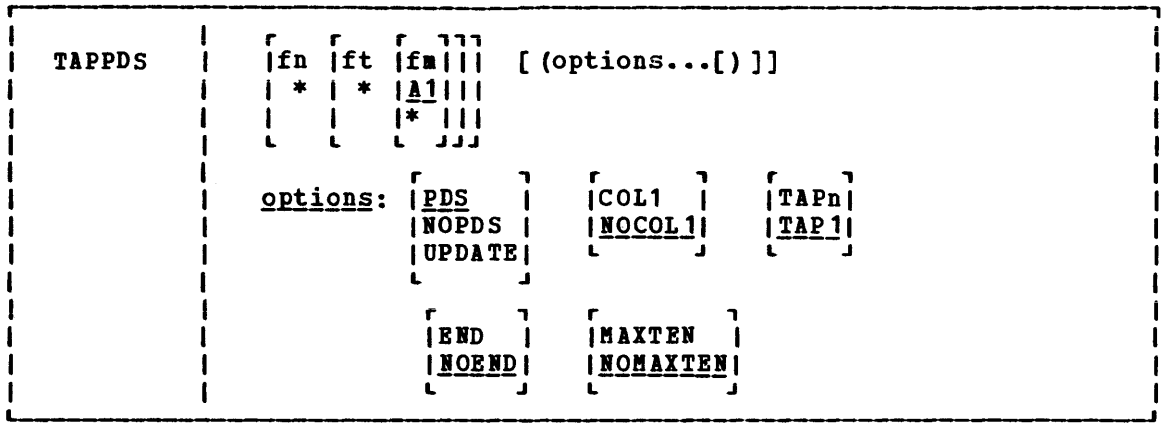
| If a tape contains large files that would not fit on disk, the tape
| load operation is terminated. To prevent this when you dump the files
| separate logical files by tape marks, then forward space to the
| appropriate file.

Other Messages and Return Codes

```
DMSTPE002D FILE (S) '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 'param' 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 'param'  RC=24
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    TRACK}    FEATURE     NOT    SUPPORTED    ON
           (TRANSLATION|DUAL-DENSITY) DEVICE 'cuu'  RC=88
```

TAPPDS

Use the TAPPDS command to create CMS disk files from either tapes in unblocked card-image format that are produced by the OS IEBPTPCH service program or from tapes in the OS IEBUPDTE service program control file format, either blocked or unblocked. Using the TAPPDS command, you can also read unloaded partitioned data sets (PDS) from a tape created by the OS IEHMOVE service program, and create a CMS disk file for each member of the data set. The tape can be unlabeled or it can contain OS standard labels. The format of the TAPPDS command is:



where:

fn is the filename of the disk file to be created. This field has meaning only if the NOPDS option is selected (that is, the tape does not contain members of a partitioned data set). If the tape does contain members of a partitioned data set (PDS), an asterisk must be specified; one file is created for each member with a filename the same as the member name. If NOPDS or UPDATE is specified, the default filename is TAPPDS. The default is assumed if the filename is omitted or coded as *.

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. The default filemode is A1 if this field is omitted or specified as an asterisk (*).

Options

If conflicting options are specified, the last one entered is used. All options, except TAPn, are ignored when unloaded 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 tape contains one file.

UPDATE — provides the "./ ADD" function of the OS IEBUPDTE service program in your CMS virtual machine. It indicates that the optional material which is in IEBUPDTE control file format is to be loaded onto disk as CMS files. The filename of the new disk file is taken from the NAME= parameter in the "./ ADD" record. The tape input file can be blocked or unblocked. All records are written onto the CMS disk in fixed-length, 80-byte format.

The END option is disabled when UPDATE is specified. The COL1 option should be used with UPDATE so that the scanning of the data starts at column one. The "./ " must appear in columns 1-3. All records that do not contain "./ " in columns 1-3 which TAPPDS encounters after an initial "./ ADD" record, are written onto disk. Conversely, if "./ ADD" is not found, the file is not created on disk. The "./ " records are not written as part of the file on disk.

An "./ ENDUP" record causes TAPPDS to close the current file and stop processing without repositioning the tape; also, a single tape mark has the same effect.

The optional 'label' in columns 3-10 of a "./ " record in an IEBUPDTE control file is not recognized by TAPPDS. The record is treated as a data record and included in the CMS disk file.

For the "./ ADD" record, if the NAME= parameter is missing or followed by a blank, TAPPDS uses the default filename "TAPPDS" for the CMS disk file. If this condition occurs more than once during command execution, then upon completion of the TAPPDS command only the last member "./ ADD" without a valid NAME= parameter is on your disk with the default filename.

COL1 — takes data from columns 1-80; column 1 contains data.

NOCOL1 — takes data from columns 2-80: column 1 contains control character information. This is the format produced by the OS IEBPTPCH service program.

TAPn
TAP1 — is the tape unit number. TAP1 is the default tape unit number, which corresponds to the virtual address 181. There are four possible values of n: TAP1 through TAP4, indicating virtual tape drives 181 through 184.

END — considers an END statement 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.

Responses

DMSTPD703I FILE 'fn ft [fm]' COPIED

The named file is copied to disk.

DMSTPD707I TEN FILES COPIED

MAXTEN was specified and ten members are copied.

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
DMSTPD110S ERROR 'nn' READING 'TAPn(cuu)'  RC=100
```

TXTLIB

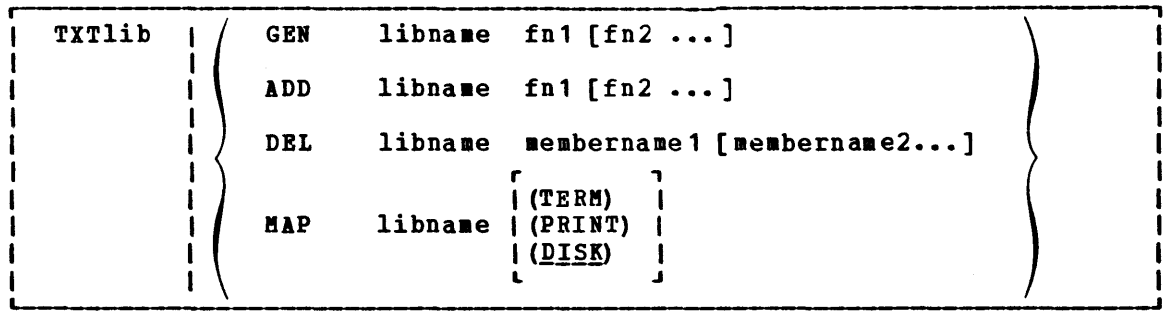
Use the TXTLIB command to update CMS text libraries. A text library is one which is to be searched for missing subroutines in LOAD and INCLUDE commands. (See Note 1.)

A text library is a file that has a filetype of TXTLIB and contains a dictionary and one or more relocatable object programs obtained from CMS files having a filetype of TEXT.

The TXTLIB command:

- Generates a text library.
- Adds to an existing text library.
- Deletes from an existing text library.
- Lists the names and aliases or entry points and control section names and the location of the TEXT files included in the text library.

The format of the TXTLIB command is:



where:

GEN libname fn1 [fn2 ...]
: generates a text library with the specified filename (libname) and a filetype of TXTLIB from the TEXT files specified by fn1 fn2... If a file exists with the identification libname TXTLIB, it is erased and a new one is created.

ADD libname fn1 [fn2 ...]
: appends the contents of the files specified by fn1 fn2... to the end of the existing library with the file identification libname TXTLIB. No checking for duplicate names, aliases, entry points, or CSECT names is performed.

DEL libname membername1 [membername2...]
: removes the text decks with member names (membername1, membername2...) from the directory of the text library, libname TXTLIB. If two members exist with the specified membername, only the first one encountered is deleted (unless the membername is given twice in the argument list).

  Deletions must be performed on the NAME or first Section Definition (SD) in the text deck. A deletion for an alias name

or subsequent entry point results in a not found message with no change to the member. DEL removes the member and all references to it.

The file is automatically compressed so that space occupied by the deleted members can be reused.

```
          ┌        ┐
          | (TERM) |
MAP libname | (PRINT) |
          | (DISK) |
          └        ┘
```

generates the file libname MAP on the primary disk. If a file already exists with the same identification, it is erased and the new file is created. The libname MAP file contains the same information as that in the dictionary of the specified text library and is in the format of a list of entry points and control section names that reside in the text library, and their location or index in the file. The options on the command line are examined to determine if the MAP is to be directed to the terminal (TERM), or the printer (PRINT), or is to remain on disk (DISK).

The MAP operand of the TXTLIB command displays a statement indicating the total number of entry points and control section names that currently exist in the TXTLIB file.

Notes

1. The total number of members in the TXTLIB file cannot exceed 1000. When this number is reached, an error message is displayed. The text library created includes all the text files entered up to (but not including) the one that caused the overflow.

2. OS Linkage Editor ENTRY, ALIAS, and NAME control statement are accepted. If a NAME statement is detected, only ALIAS and NAME 'names' are included in the dictionary for that text deck. Deletions must be performed on the NAME 'name'. The total number of ALIAS names cannot exceed sixteen names per text deck.

3. Unlike OS STEPLIB entries, CMS TXTLIB members are not fully link-edited. The loader, for either an explicit or dynamic load, attempts to resolve all external references. For a dynamic load, if all VCONs cannot be resolved within a member, an incorrect entry point might be returned. You should explicitly load those subroutines by either CMS LOAD and INCLUDE commands or by a VCON in the program.

Response

xxx ENTRIES IN LIBRARY

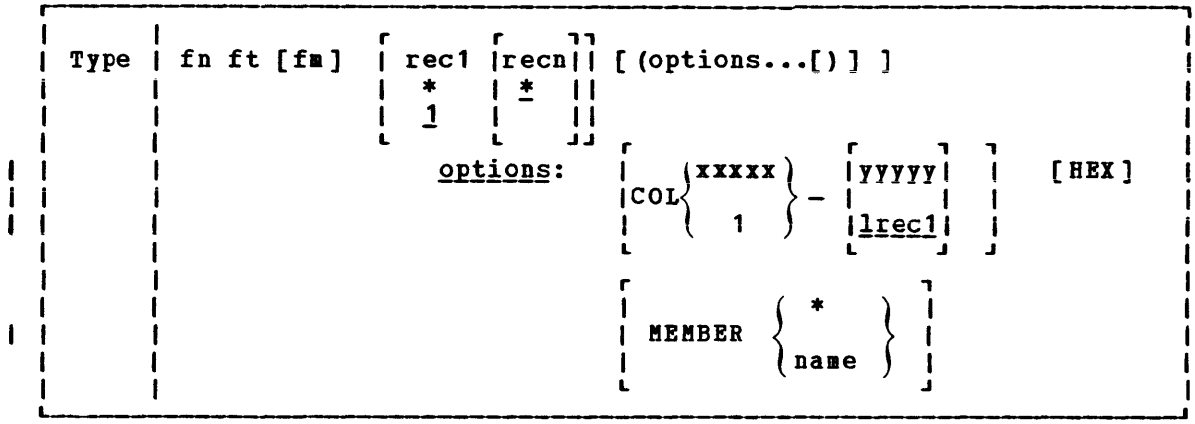
When TXTLIB is issued, the contents of the dictionary of the specified text library are displayed at the terminal. The number of entries in the text library (xxx) is displayed at the terminal when the TXTLIB MAP command is issued.

Other Messages and Return Codes

```
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
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
DMSLBT104S ERROR 'nn' READING FILE 'fn ft fm' FROM DISK  RC=100
DMSLBT105S ERROR 'nn' WRITING FILE 'fn ft fm' ON DISK  RC=100
DMSLBT106S NUMBER OF MEMBER NAMES EXCEEDS MAX  'nnnn'.  FILE 'fn ft' NOT
           ADDED  RC=88
DMSLBT213W LIBRARY 'fn ft fm' NOT CREATED  RC=4
```

TYPE

Use the TYPE command to display all or part of a specified file at the terminal in either EBCDIC or the hexadecimal representation of the EBCDIC code. The format of the TYPE command is:



where:

fn is the filename of the file to be displayed. This field must be specified.

ft is the filetype of the file to be displayed. This field must be specified.

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. In the case of files with duplicate filename and filetype, only 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, the file length is assumed. 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 not specified or is entered as an asterisk, display continues until end of file is reached.

Options

COL xxxxx yyyyy
: displays only certain positions of each record. xxxxx specifies the beginning position and yyyyy the ending position of the field within the record which is to be displayed. If a field is 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 the file specified is a library, 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.

Response

The file is displayed at the terminal according to the given specifications.

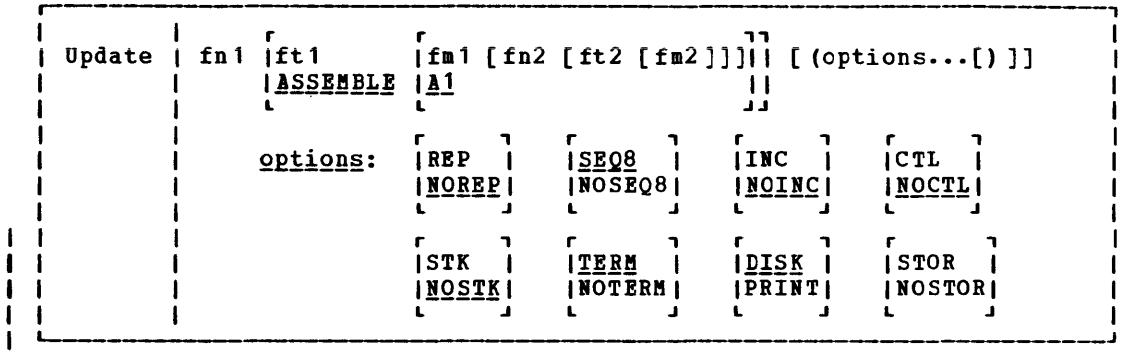
Error Messages and Return Codes

```
DMSTYP002E FILE 'fn ft fm' NOT FOUND  RC=28
DMSTYP003E INVALID OPTION 'option'  RC=24
DMSTYP005E INVALID 'option' SPECIFIED  RC=24
DMSTYP009E COLUMN 'col' EXCEEDS RECORD LENGTH  RC=24
DMSTYP013E MEMBER 'name' NOT FOUND IN LIBRARY  RC=32
DMSTYP029E INVALID  PARAMETER 'param'  [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 source files should be stored as 80-character card-image files with sequence fields in record positions 73 to 80. The UPDATE command accepts a source input file and one or more files containing update control statements and update source records. The UPDATE command 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.

Updates may be applied either permanently (that is, the updated output file replaces the source input file), or temporarily, in which case the updated output file has the name '$fn', where 'fn' is the file name of the input source file. The format of the UPDATE command is:



where:

fn1 ft1 fm1
: is the filename, filetype, and filemode of the source input file. If the filemode or filetype is omitted, 'A1' and 'ASSEMBLE' are assumed, respectively.

fn2 ft2 fm2
: is the filename, filetype, and filemode of the file containing the update control statements and updated source records, or, if the CTL option is specified, specifies the filename, filetype, and filemode of the update control file to be used for a multiple update. The defaults are fn1 UPDATE A1 if NOCTL is in effect, and fn1 CNTRL A1 if CTL is specified.

Options

REP
: replaces with the source input file with the updated source file.

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 8-digit sequence number on every record of source input.

NOSEQ8 specifies that columns 73-75 contain a 3 character label field, and that the sequence number is a 5-digit value in columns 76-80.

INC puts sequence numbers in columns 73 through 80 of each updated record inserted from the update file.

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 "Control Statement Formats" section that follows).

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 resulting from a multiple update at your CMS console. This information can then be read by a CMS EXEC procedure. STK is used only with the CTL option.

NOSTK specifies that no external communication of the multiple update results is desired.

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 displaying of warning messages at the terminal. However, error messages which terminate the entire update procedure are displayed at the terminal.

DISK places the update log file on disk. This file has a file identification "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 on disk. This option is meaningful only when used in conjunction with CTL option since the benefit of increased processing speed is realized when processing multiple updates. STOR is the default when CTL is specified.

| NCSTOR specifies that no updating is to take place in storage. NOSTOR is the default when performing single updates or when CTL is omitted from the command line.

Control Statements

The UPDATE control statements let you insert, delete and replace source records, as well as resequence the output file. 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 a maximum of 6 additional, blank-delimited fields. Control statement data must not extend beyond column 50. 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.

Any sequence fields in the update control statements are ignored; if the NOINC option is specified, all sequence fields in the update file are ignored, including those on inserted records. If the INC option is specified, sequence fields for the inserted records are either generated by UPDATE, if the dollar-sign ($) delimiter is used, or are included intact from the update file if the dollar sign ($) is not used.

Changes are made sequentially in a single pass through the input and update files; 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

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). If this statement is included in the update file, it must be the first control statement. The format of the Sequence control statement is:

```
| ./  S  [seqstrt  [seqincr  [label]]]  |
```

where:

seqstrt — 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 — a one- to eight-digit numeric field specifying the decimal increment for resequencing the output file. The default is the 'seqstrt' value.

label — a 3-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).

An error is indicated if any valid control statement precedes the ./ S statement in the update file, and the resequence operation is suppressed.

Each source record is resequenced in columns 73-80 as it is written onto the output file. Both unchanged records from the input file and records inserted from the update file are resequenced.

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 source input record following which the insertion is to be made.

$ — optional delimiter indicating that the inserted records are to be sequenced incrementally.

seqstrt — a one- to eight-digit numeric field specifying the first decimal number to be used for sequencing the inserted records.

seqincr — 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, non-zero 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 maximun, 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, are deleted from the output file, up to and including the seqno2 record. 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 a one- to eight-digit numeric field specifying the first decimal number to be used for sequencing the substituted records.

seqincr 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 following 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, such that 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 Statements --The format of the Comment statement is:

```
| ./  *  [comment]                                    |
```

where:

* indicates that this is a comment statement, and is to be ignored, except that it is copied into the log file.

Summary of Input and Output Files for the UPDATE Command

The following sections describe the files used and created by the UPDATE command, and the placement of the files created.

INPUT FILES WHEN A SINGLE UPDATE IS TO BE APPLIED: When the CTL option is not specified in the UPDATE command line, only one update is applied to the source file. The input files are:

- The source file, which is to be updated. The filename of this file must be specified in the command line. The filetype and filemode default to ASSEMBLE and A1, respectively, unless overridden by the command line.

- The update file, whose control statements have been described in the preceding section. The filename of this file defaults to the filename of the source file, and the filetype and filemode default to UPDATE and A1, respectively. All three may be overridden by the command line.

OUTPUT FILES WHEN A SINGLE UPDATE IS APPLIED: When a single update is applied to the source file, the following output files are created:

- An updated source file is created. "$fn" becomes the name of this file, where "fn" is the filename of the original source file, unless the REP option is specified. When the REP option is specified, the filename of this file becomes "fn". (For exceptions, see the "Warning and Error Handling" section that follows.)

- An update log, showing all transactions and errors, is created. The filename of the file is the filename of the original source file, and the filetype of this file is UPDLOG. Note, however, that if the PRINT option is specified with the command line, then the update log is printed directly on the virtual spooled printer, and no disk file is created.

INPUT FILES WHEN MULTILEVEL UPDATES ARE APPLIED: When the CTL option is specified on the command line, multilevel updates are applied to the source file. In this case, the following files are input to the UPDATE command:

- A source file, specified in exactly the same way as the source file for a single update.

- A control file that controls what updates are applied, and the order in which they are to be applied. The filename of this file defaults to the filename of the source file, and the filetype and filemode default to CNTRL and A1, respectively. All three may be overridden by the command line. This file contains, in its control statements, pointers to update files, PTF files, and auxiliary files (these are described in "The CTL Option" section).

- One or more update files, as specified by the control file. The filename of these files is the same as the filename of the source file. The filetype of these files is "UPDTxxxx".

- Auxiliary files, as specified by the control file. The filename of these files is the same as the filename of the source file. The filetype of these files is "AUXxxxx". The filetype can be specified as either 'AUXnnnn' or 'nnnn AUX'. If you use the second form, the first three characters may not be 'AUX'. The auxiliary files contain additional control statements pointing to PTF files. The format of the auxiliary files is described in a later section, "The CTL Option."

- PTF files, as specified by either the control file or the auxiliary files. The filename of these files is the same as the filename of the source file. The filetype is specified in full by the control file or the auxiliary file. In format, these files are identical to ordinary update files.

OUTPUT FILES WHEN MULTILEVEL UPDATES ARE APPLIED: When the CTL option is specified, the following output files are created by the UPDATE command:

- An updated source file, as in the case of a single update.

- An update log, as in the case of a single update.

- An UPDATES file. This file has the filename of the original source file, and a filetype of UPDATES. It contains summary information about which updates were applied to the file, and is intended to be concatenated onto the assembly text deck for documentation and information purposes.

- Although not a disk file, additional "output" is produced in the form of lines placed in the terminal read stack, for interrogation by an EXEC file which may have invoked the UPDATE command. These lines are placed there only if the STK option is specified.

DISK MODE OF OUTPUT FILES: If there are several read/write disks 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):

- If the disk on which the original source file resides is read/write, then the output files are placed on that disk.

- 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.

- Otherwise, the output files are place on the primary read/write disk (the A-disk).

The CTL Option

If the NOCTL option is specified or defaulted, UPDATE processes one input file and one update file to produce an updated source output file and an update log file containing a record of what changes were made. This mode of operation is suitable for testing modifications prior to incorporating them in the base source code, providing that only one set of changes has to be tested at a time. If, for any reason, more than one set of changes is outstanding against a single source input file, the difficulties in managing that base code can multiply very rapidly. For this reason, UPDATE provides the CTL option, which has a multilevel update control and management scheme developed for updating VM/370 distributed source code, and may be used wherever its advantages are felt.

The major components of the multilevel update scheme are:

- A set of base source code which is not permanently changed.
- A set of update files for each source file that must be applied in a specific order.
- One or more CNTRL files that describe the order or priority of updates to be applied to each source file.
- Optionally, one or more auxiliary control files, each pertaining to a specific source file.

An integral part of the multilevel update scheme is a naming convention for the update files themselves, and for any TEXT files produced by assembling or compiling the updated output files. In normal usage, any update file has the filename of the source file to which it applies and the filetype of UPDATE. When the CTL option is used to invoke the multilevel update controls, the filename usage becomes a requirement, such that the update files must have the filename of the source file to which they apply, but the filetypes are modified to distinguish between separate update levels. The filetype for an update file is constructed from UPDT plus a one- to four-character update identifier. For example, if the command

```
UPDATE DMSUPD ASSEMBLE A1 X4 CNTRL A1 (CTL
```

is issued, the source file is DMSUPD ASSEMBLE A1 and the control file is X4 CNTRL A1. Assume that the control file contains three update files, named "DMSUPD UPDT750", "DMSUPD UPDTX4", and "DMSUPD UPDT009." The CNTRL file specifies which update files are to be applied to the source file and in what order they are to be applied, on the basis of the update identifier. Another identification parameter, the update level identifier, is used when naming a TEXT file produced from the updated source file. The update level identifier is specified by the CNTRL file and is associated with a specific update identifier, also in the CNTRL file. For example, a file named X4 CNTRL, to apply the above mentioned updates to DMSUPD ASSEMBLE, might appear as follows:

```
00D  MACS  DMSLIB  SYSLIB
X4D  UPDTX4
75X  UPDT750
009X UPDT009
```

This control file applies the updates DMSUPD UPDT009, DMSUPD UPDT750, and DMSUPD UPDTX4, in that order, to the file DMSUPD ASSEMBLE. The updates are applied in reverse order as they appear in the CNTRL file, that is, the lowest level of update is at the bottom of the file, and the highest level update is at the top. As the CNTRL file and update files are processed, the UPDATE command displays the following message at the terminal:

```
DMSUPD178I UPDATING ['fn ft fm'] WITH 'fn ft fm'
```

for each update file which is applied to the source input during the multilevel update; the bracketed expression is displayed only for the first update.

In the above example, the fields X4D, 75X, and 009X are the update level identifiers, associated with the UPDTX4, UPDT750, and UPDT009 update identifiers, respectively. According to the naming convention for VM/370 TEXT files, the result of assembling the updated $DMSUPD ASSEMBLE file would be named DMSUPD TXTX4D, where the X4D is the update level identifier of the highest-level update applied. The TXT portion of the filetype indicates that this is a TEXT file, but allows up to a five-character update level identifier.

The STK/NOSTK Options

The STK option is provided for use with the multilevel update invoked via the CTL option, primarily for communication with CMS EXEC procedures which invoke UPDATE. If the CTL and STK options are specified, UPDATE places two lines of data in the CMS terminal read stack, as follows:

```
first line  =  * update level identifier

second line =  * library list from 'MACS' record
```

These lines are placed in the terminal read stack via the CMS ATTN function, and are available to an invoking EXEC procedure via the EXEC control words &READ ARGS or &READ VARS. The first line, the update level identifier, is the level identifier of the highest level update applied; this is the TEXT file filetype-modifier used by the VM/370 update procedures. The second line consists of the list of libraries specified on the MACS record in the CNTRL file. The library search order for an assembly or compilation can be established by issuing the GLOBAL command using the library list returned.

If the NOSTK option is used with the multilevel update, no data is made available to external procedures and the update level identifier has no meaning.

Example of Multilevel Update

If the command

```
update proga assemble * proga cntrl * (ctl stk)
```

is issued and the contents of the PROGA CNTRL file are :

```
* THIS IS AN EXAMPLE OF A CONTROL FILE
00D MACS MYLIB SYSLIB
* FIX FOR SAVING ALL REGISTERS
00A UPDTLVL4
PTF A7300DMS
* AUX FILE CONTAINING UPDATES FOR B1 FEATURE
00B AUXLVL3
```

and the contents of the PROGA AUXLVL3 file are :

```
* PTF A2330DMS CONTAINS FUNCTIONAL CODE FOR B1
PTF A2330DMS
* PTF A0915DMS CONTAINS ERROR MESSAGES FOR B1
PTF A0915DMS
```

The files listed below are used to update PROGA ASSEMBLE in the order indicated.

Filename	Filetype
PROGA	A0915DMS
PROGA	A2330DMS
PROGA	A7300DMS
PROGA	UPDTLVL4

The resultant output file $PROGA ASSEMBLE contains the updates from the four files listed. In addition, two lines are placed in front of the CMS terminal read stack:

```
* 00A
* MYLIB SYSLIB
```

If the UPDATE command shown was issued from an EXEC procedure and followed by the EXEC commands:

```
&READ VARS &I1 &I2
&READ VARS &I3 &I4 &I5
```

then

```
&I1 = *
&I2 = 00A
&I3 = *
&I4 = MYLIB
&I5 = SYSLIB
```

In this example, the UPDATE command was entered from the terminal and the stacked lines are ignored by CMS.

Warning and Error Handling

The UPDATE command detects a number of invalid requests, and decides whether they should be treated as warning situations or as errors. The following is a general description of the handling of these situations, and the return codes associated with each.

SEQUENCING AND UPDATE CONTROL CARD ERRORS: These errors are treated as "warning situations." That is, a warning message is generated, and processing continues. The warning messages are printed in the update log, and are displayed on the terminal unless the NOTERM option is specified in the command line. The errors are:

- Input sequencing errors (return code = 4). The input source file contains sequence errors (sequence numbers in nonascending order).
- Output sequencing errors (return code = 8). The updating procedure introduces new sequencing errors into the output source file.
- Invalid update control statements (return code = 12). The update file contains invalid control statements. Erroneous statements in control files cause the update procedure to terminate.

If more than one such error is detected, the UPDATE command returns the highest return code (4, 8, or 12) encountered.

If any such error is detected, the REP option, if specified, is ignored, and the update source file retains the filename "$fn", as if NOREP was in effect.

OTHER ERRORS: Other errors are invalid control file statements, invalid file formats, and disk input/output errors. The UPDATE command processing is terminated as soon as the error is detected. The return code is always 20 or greater.

If any such error is detected, the update file is left with the filename UPDATE and the filetype CMSUT1, so that you may examine or otherwise make use of it. This file must be erased before the UPDATE command can be invoked again.

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' OPTION 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 which 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 has a return code of 12. Such errors include invalid update file control statements, and missing 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.

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=24
DMSUPD037E DISK 'A' 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 'A' NOT ACCESSED  RC=36
DMSUPD070E INVALID PARAMETER 'param'  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=4
DMSUPD210W INPUT FILE SEQUENCE ERROR: '.........' TO '.........'  RC=4
```

IMMEDIATE COMMANDS

An IMMEDIATE command is issued after an Attention interrupt is given to CMS by pressing the Attention key (or its equivalent). Such commands are processed immediately upon entry from the terminal or on being 'stacked' by an EXEC procedure. Any program execution in progress is suspended until the immediate command is processed. The commands are HB (halt batch execution), HO (halt SVC tracing), HT (halt typing), HX (halt execution), RO (resume tracing), RT (resume typing), and SO (suspend tracing temporarily).

HB

Use the HB command to stop the execution of a CMS Batch virtual machine at the end of the current job. If the virtual machine is running in disconnect mode, it must be reconnected. Press the Attention Key to stop program execution and enter the command. 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 out the CMS Batch virtual machine.

```
┌──────────────────────────────────────────────────────────────────┐
│ HB │                                                             │
└──────────────────────────────────────────────────────────────────┘
```

Responses

None.

HO

Use the HO command during the execution of a command or one of your programs to stop the recording of trace information. In order for the HO command to be recognized, it must be entered after you stop program execution by an Attention interrupt. 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 │                                                             │
└──────────────────────────────────────────────────────────────────┘
```

Responses

None.

HT

Use the HT command to suppress all terminal output generated by any CMS command or your program that is currently executing. In order for the HT command to be recognized when entered, an Attention interrupt must be simulated by pressing the Attention key or its equivalent. Program execution continues. With typewriter terminals, the Attention key unlocks the keyboard to accept your HT command. With display terminals, you enter the HT command in the input area and then press the Enter key. When the Ready message is displayed, normal terminal output resumes. The format of the HT command is:

```
| HT |
```

Responses

None.

HX

Use the HX command to stop the execution of any CMS command or one of your programs under CMS, close any open files or I/O devices, and return to the CMS command environment. In order for the HX command to be recognized, it must be issued after you stop program execution by an Attention interrupt. The HX command is executed when the next SVC or I/O interrupt occurs. All terminal output generated before HX is executed is displayed before the command is executed. The format of the HX command is:

```
| HX |
```

Responses

None.

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. In order for the RO command to be recognized, it must be entered after you stop program execution with an Attenion interrupt. 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 |
```

Responses

None.

RT

Use the RT command to restore terminal displaying from an executing CMS command or one of your programs that was previously suppressed by the HT command. In order for the RT command to be recognized when entered, an Attention interrupt must be simulated by pressing the Attention key or it equivalent. Program execution continues, and displaying continues from the current point of execution in the program. Any console 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. The format of the RT command is:

```
| RT |
```

Responses

None.

SO

Use the SO command during the execution of a command or one of your programs to temporarily suspend the recording of trace information. Tracing resumes if you issue the RO command at a later time. In order for the SO command to be recognized, it must be entered after you stop program execution with an Attention interrupt. 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 |                                                             |
└──────────────────────────────────────────────────────────────────┘
```

Responses

None.

CP Command Privilege Classes

The CP commands are divided into eight privilege classes, each class representing a different type of user. Each user is assigned, as part of his entry in the VM/370 directory, one or more privilege classes. Figure 33 shows the function of each class. The footnotes following the privilege class indicate where the commands of that particular privilege class are described. Figure 34 shows which commands (and which operands, if the command varies for different privilege classes) are associated with each class. Only class G commands and class Any commands are included in this publication.

Class	User and Function
A[1]	Primary System Operator: The class A user controls the VM/370 system. Class A is assigned to the user at the VM/370 system console at IPL time. The primary system operator is responsible for the availability of the VM/370 system and its communication lines and resources. In addition, the class A user controls system accounting, broadcast messages, virtual machine performance options and other command operands that affect the overall performance of VM/370. Note: The class A system operator who is automatically logged on during CP initialization is designated as the Primary System Operator.
B[1]	System Resource Operator: The class B user controls all the real resources of the VM/370 system, except those controlled by the primary system operator and spooling operator.
C[1,2]	System Programmer: The class C user updates certain functions of the VM/370 system.
D[1]	Spooling Operator: The class D user controls spool data files and specific functions of the system's unit record equipment.
E[1,2]	System Analyst: The class E user examines and saves certain data in the VM/370 storage area.
F[1,3]	Service Representative: The class F user obtains and examines in detail, certain data about input and output devices connected to the VM/370 system.
G[4]	General User: The class G user controls functions associated with the execution of his virtual machine.
Any[1,4]	Any User: The class Any user has limited use of VM/370 to gain initial access to the VM/370 system.
H	Reserved for IBM use.

[1]Described in the VM/370: Operator's Guide.
[2]Described in the VM/370: System Programmer's Guide.
[3]Described in the VM/370: OLTSEP and Error Recording Guide.
[4]Described in this publication.

Figure 33. CP Privilege Class Descriptions

Class	Commands	Operands
any	*	
	#CP	
	CP	
	DIAL	
	DISCONN	
	LOGOFF	
	LOGON	
	MESSAGE	
	SLEEP	
A	ACNT	
	DISABLE	
	ENABLE	
	FORCE	
	HALT	
	LOCK	
	MESSAGE	ALL
	MONITOR	DISPLAY
		ENABLE
		INTERVAL
		START
		STOP
	NETWORK	DISABLE
		DISPLAY
		DUMP
		ENABLE
		HALT
		LOAD
		POLLDLAY
		QUERY
		SHUTDOWN
		VARY
	QUERY	LOGMSG
		NAMES
		PRIORITY
		SASSIST
		userid
		USERS
	SET	FAVORED
		PRIORITY
		RESERVE
		SASSIST
	SHUTDOWN	
	UNLOCK	
	WARNING	
B	ATTACH	
	ATTACH	CHANNEL
	DETACH	
	DETACH	CHANNEL
	DISABLE	
	ENABLE	
	MESSAGE	ALL
	NETWORK	DISABLE
		DISPLAY
		DUMP
		ENABLE
		LOAD
		POLLDLAY
		QUERY
		SHUTDOWN
		VARY
	QUERY	ALL
		DASD
		DUMP
		GRAF
		LINES
		LOGMSG
		NAMES
		raddr
		SPOOL
		STORAGE
		SYSTEM
		TAPES
		TDSK
		UR
		userid
		USERS
	SET	DUMP
		LOGMSG
	VARY	
	WARNING	
C	QUERY	LOGMSG
		NAMES
		userid
		USERS
	STCP	
D	BACKSPAC	
	CHANGE	
	DRAIN	
	FLUSH	
	FREE	
	HOLD	
	LOADBUF	
	ORDER	
	PURGE	
	QUERY	FILES
		HOLD
		LOGMSG
		NAMES
		PRINTER
		PUNCH
		READER
		UR
		userid
		USERS
	REPEAT	
	SPACE	
	START	
	TRANSFER	
E	DCP	
	DMCP	
	INDICATE	I/O
		LOAD
		PAGING
		QUEUES
		USER

Figure 34. Commands Accepted from Each User Class (Part 1 of 2)

Class	Commands	Operands
	LOCATE	
	MONITOR	DISPLAY
		ENABLE
		INTERVAL
		START
		STOP
	QUERY	LOGMSG
		NAMES
		PAGING
		PRIORITY
		SASSIST
		userid
		USERS
	SAVESYS	
F	NETWORK	TRACE
	QUERY	LOGMSG
		NAMES
		userid
		USERS
	SET	RECORD
		MODE
G	ADSTOP	
	ATTN	
	BEGIN	
	CHANGE	
	CLOSE	
	COUPLE	
	DEFINE	
	DETACH	
	DISPLAY	
	DUMP	
	ECHO	
	EXTERNAL	
	INDICATE	LOAD
		USER
	IPL	
	LINK	
	LOADVFCB	
	NOTREADY	
	ORDER	
	PURGE	
	QUERY	CHANNELS
		FILES

Class	Commands	Operands
		LINKS
		LOGMSG
		NAMES
		PFnn
		PRINTER
		PUNCH
		READER
		SET
		TERMINAL
		TIME
		userid
		USERS
		VIRTUAL
	READY	
	REQUEST	
	RESET	
	REWIND	
	SET	ACNT
		ASSIST
		ECMODE
		EMSG
		IMSG
		ISAM
		LINEDIT
		MSG
		NOTRANS
		PAGEX
		PFnn
		PFnn COPY
		PFnn TAB
		RUN
		TIMER
		WNG
	SPOOL	
	STORE	
	SYSTEM	
	TAG	DEV
		FILE
		QUERY
	TERMINAL	
	TRACE	
	TRANSFER	

Figure 34. Commands Accepted from Each User Class (Part 2 of 2)

CP Command Summary

This section contains descriptions of the commands acceptable in the control program environment. Figure 35 presents an alphabetical list of the commands, the privilege classes which may execute the command, and a brief statement about the use of each command.

Command	Privilege Class	Usage
*	any	Annotate the console sheet.
#CP	any	Execute a CP command while remaining in the virtual machine environment.
ACNT	A	Create accounting records for logged on users and reset accounting data.
ADSTOP	G	Halt execution at a specific virtual machine instruction address.
ATTACH	B	Attach a real device to a virtual machine. Attach a DASD device for CP control. Dedicate all devices on a particular channel to a virtual machine.
ATTN	G	Make an attention interrupt pending for the virtual machine console.
BACKSPAC	D	Restart or reposition the output of a unit record spooling device.
BEGIN	G	Continue or resume execution of the virtual machine at either a specific storage location or at the address in the current PSW.
CHANGE	D,G	Alter one or more attributes of a closed spool file.
CLOSE	G	Terminate spooling operations on a virtual card reader, punch, printer, or console.
COUPLE	G	Connect channel-to-channel adapters.
CP	any	Ignored.
DCP	E	Display real storage at terminal.
DEFINE	G	Reconfigure your virtual machine.
DETACH	B B B G	Disconnect a real device from a virtual machine. Detach a DASD device from CP. Detach a channel from a specific user. Detach a virtual device from a virtual machine.
DIAL	any	Connect a terminal or display device to the virtual machine's virtual communication line.
DISABLE	A,B	Disable 2701/2702/2703 and 3270 communication lines.

Figure 35. CP Command Summary (Part 1 of 4)

Command	Privilege Class	Usage
DISCONN	any	Disconnect your terminal from your virtual machine.
DISPLAY	G	Display virtual storage on your terminal.
DMCP	E	Dump the specified real storage location on your virtual printer.
DRAIN	D	Halt operations of specified spool devices upon completion of current operation.
DUMP	G	Print on virtual printer: virtual PSW, general registers, floating-point registers, storage keys, and contents of specified virtual storage locations.
ECHO	G	Test terminal hardware by redisplaying data entered at the terminal.
ENABLE	A,B	Enable communication lines.
EXTERNAL	G	Simulate an external interrupt for a virtual machine and return control to that machine.
FLUSH	D	Cancel the current file being printed or punched on a specific real unit record device.
FORCE	A	Cause logoff of a specific user.
FREE	D	Remove spool HOLD status.
HALT	A	Terminate the active channel program on specified real device.
HOLD	D	Defer real spooled output of a particular user.
INDICATE	E,G	Indicate resource utilization and contention.
IPL	G	Simulate IPL for a virtual machine.
LINK	G	Provide access to a specific DASD device by a virtual machine.
LOADBUF	D	Load real UCS/UCSB or FCB printer buffers.
LOADVFCB	G	Load virtual forms control buffer for a virtual 3211 printer.
LOCATE	E	Find CP control blocks.
LOCK	A	Bring virtual pages into real storage and lock them; thus excluding them from future paging.
LOGOFF	any	Disable access to CP.
LOGON	any	Provide access to CP.
MESSAGE	A,B,any	Transmit messages to other users.

Figure 35. CP Command Summary (Part 2 of 4)

Command	Privilege Class	Usage
MONITOR	A,E	Trace events of the real machine and record system performance data.
NETWORK	A,B,F	Load, dump, trace and control the operation of the 3704/3705 control program.
NOTREADY	G	Simulate "not ready" for a device to a virtual machine.
ORDER	D,G	Rearrange closed spool files in a specific order.
PURGE	D,G	Remove closed spool file from system.
QUERY	A,B,C,D,E,F,G	Request information about machine configuration and system status.
READY	G	Simulate device end interrupt for a virtual device.
REPEAT	D	Repeat (a specified number of times) printing or punching of a specific real spool output file.
REQUEST	G	Make an attention interrupt pending for the virtual machine console.
RESET	G	Clear and reset all pending interrupts for a specified virtual device and reset all error conditions.
REWIND	G	Rewind (to load point) a tape and ready a tape unit.
SAVESYS	E	Save virtual machine storage contents, registers, and PSW.
SET	A,B,F,G	Operator--establish system parameters. User--control various functions within the virtual machine.
SHUTDOWN	A	Terminate all VM/370 functions and checkpoint CP system for warm start.
SLEEP	any	Place virtual machine in dormant state.
SPACE	D	Force single spacing on printer.
SPOOL	G	Alter spooling control options; direct a file to another virtual machine or to a remote location via the RSCS virtual machine.
START	D	Start spooling device after draining or changing output classes.
STCP	C	Change the contents of real storage.
STORE	G	Alter specified virtual storage and registers.

Figure 35. CP Command Summary (Part 3 of 4)

Command	Privilege Class	Usage
SYSTEM	G	Simulates RESET, CLEAR STORAGE and RESTART buttons on a real system console.
TAG	G	Specify variable information to be associated with a spool file or output unit record device.
		Interrogate the current TAG text setting of a given spool file or output unit record device.
TERMINAL	G	Define or redefine the input and attention handling characteristics of your virtual console.
TRACE	G	Trace specified virtual machine activity at your terminal, spooled printer, or both.
TRANSFER	D,G	Transfer input files to or get input files from a specified user's virtual card reader.
UNLOCK	A	Unlock previously locked page frames.
VARY	B	Mark a device unavailable or available.
WARNING	A,B	Transmit a high priority message to a specified user or to all users.

Figure 35. CP Command Summary (Part 4 of 4)

*

Privilege Class: Any

Use * to annotate the terminal console sheet or terminal display screen data with a comment. This commentary also appears in the virtual console spool file (if the console spooling function is invoked for the virtual machine). The format of the * (comment) command is:

*	anycomment

Responses

None.

#CP

Privilege Class: Any

Use the #CP command to execute a CP command while in a virtual console read environment without first signalling attention to get to the CP environment. The format of the #CP command is:

```
| #CP | [commandline1 [#commandline2 #...]]
```

where:

commandline specifies the name, operands, and options for the CP command or commands you want to issue. You must precede the first commandline with at least one blank.

The pound sign (#) shown in the format above represents the logical line end symbol currently in effect for your virtual machine. If you have redefined the logical line end symbol, #CP is an invalid command; you must substitute your line end symbol for the pound sign when using this command.

Usage:

For the command to operate, the following conditions must be met:

- The virtual machine must be running with SET LINEDIT ON (a default).
- The first three characters of the edited line must be #CP (upper or lowercase) with the "#" representing the logical line end character currently defined.
- At least one blank must separate the #CP from any command line. Do not use attention interruption in any part of the line or to enter the line.

You can enter multiple command lines as operands of the #CP command, but you must separate each command line by the logical line end (#) character. If you enter only #CP with no operands, the virtual machine enters the CP environment. CP cancels the virtual machine's console READ by returning a unit exception status for the virtual console. The virtual operating system then reissues the console READ to allow you to key in the appropriate response to a previous message from that machine's operating system.

The following examples show several ways to use #CP:

Command	System Action
#CP	Your virtual machine enters CP environment.
#CP query files	QUERY command executed.
#CP query files#query users	Two separate QUERY commands executed.
data entered¢#CP msg op is tape available	"Data entered" is ignored. You send a message to the operator.
#CP data entered	You enter CP environment and CP interprets "data entered" as an invalid operand.
data entered¢#CP	"Data entered" is ignored. You enter CP environment.
#CP query files¢data entered	QUERY command is not executed; console input (data entered) passes to the virtual machine.

Responses:

If you enter #CP without a commandline, you receive this message:

```
CP
```

If you enter #CP with commandlines, you receive the responses appropriate to the individual commands you entered.

ADSTOP

Privilege Class: G

Use the ADSTOP command to halt execution at a virtual instruction address. Execution halts when the instruction at the specified address is the next instruction to be executed.

When execution halts, the CP command mode is entered and a message is displayed. At this point, you may invoke other CP debugging commands. To resume operation of the virtual machine, issue the BEGIN command. Once an ADSTOP location is set, it may be removed by one of the following:

- Reaching the virtual storage location specified in the ADSTOP command
- Performing a virtual IPL or SYSTEM RESET
- Issuing the ADSTOP OFF command
- Specifying a different location with a new ADSTOP hexloc command

The format of the ADSTOP command is:

where:

hexloc is the hexadecimal representation of the virtual instruction address where execution is to be halted. The specified address cannot be in a storage segment shared with other users, since the ADSTOP function modifies storage.

OFF cancels any previous ADSTOP setting.

Notes:

1. Since the ADSTOP function modifies storage (by placing a CP SVC X'B3' at the specified location) your program should not examine the two bytes at the instruction address. CP does not verify that the location specified contains a valid CPU instruction.
2. Address stops may not be set in an OS/VS or DOS/VS virtual machine's virtual storage; address stops may only be set in the virtual equals real partitions or regions of those virtual machines.

3. If the SVC handling portion of the virtual machine assist feature is enabled on your virtual machine, CP turns it off when an address stop is set. After the address stop is removed, CP returns the assist feature SVC handling to its previous status.

Response

ADSTOP AT xxxxxx

The instruction whose address is xxxxxx is the next instruction scheduled for execution. The virtual machine is in a stopped state. Any CP command (including an ADSTOP command to set the next address stop) can be issued. Enter the CP command BEGIN to resume execution at the instruction location xxxxxx, or at any other location desired.

ATTN

Privilege Class: G

Use the ATTN command to make an "attention" interrupt pending at your virtual console. The format of the ATTN command is:

The REQUEST command performs the same function as ATTN and the two commands can be used interchangeably.

The BEGIN command is not required after you issue ATTN.

Responses

None.

BEGIN

Privilege Class: G

Use the BEGIN command to continue or resume execution in the virtual machine at either a specified storage location or the location pointed to be the virtual machine's current PSW. The format of the BEGIN command is:

```
| Begin | [hexloc]
```

where:

hexloc is the hexadecimal storage location where execution is to begin. When BEGIN is issued without hexloc, execution begins at the storage address pointed to by the current virtual machine PSW. Unless the PSW has been altered since the CP command mode was given control, the location stored in the PSW is the location where the virtual machine stopped.

When BEGIN is issued with a storage location specified, execution begins at the specified storage location. The specified address replaces the instruction address in the PSW, then the PSW is loaded.

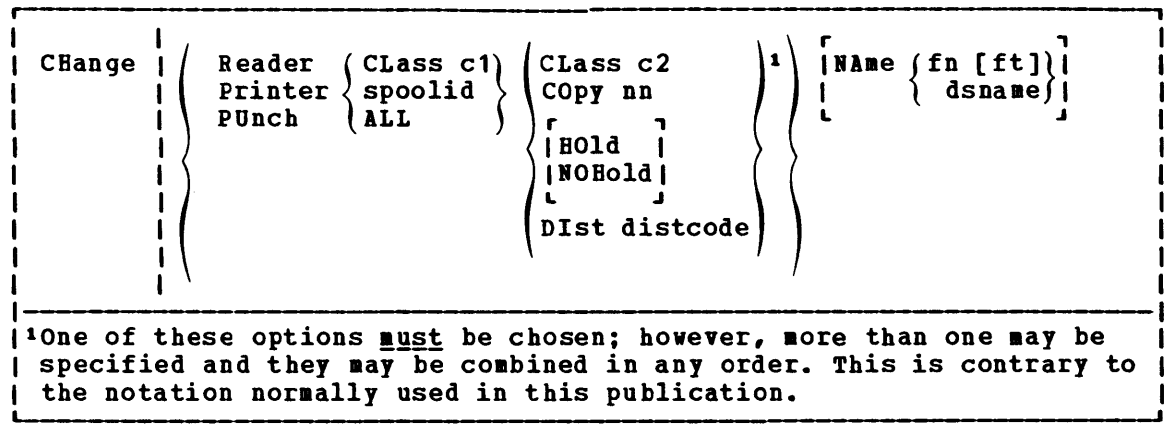
Responses

None. The virtual machine begins execution.

CHANGE

Privilege Class: G

Use the CHANGE command to alter one or more of the external attributes of a closed spool file or files. Issue the QUERY command to determine the current attributes of the file. In order to change an output file, the file must have been closed but not yet selected for printing or punching. An input (READER) file can be changed at any time before it is opened, that is, before the first read is issued for the file. The format of the CHANGE command is:



where:

READER
RDR — changes the reader spool file.

PRINTER
PRT — changes the printer spool file.

PUNCH
PCH — changes the punch spool file.

CLASS c1 — designates an existing class. The class, c1, is a one-character alphameric field from A to Z or from 0-9. Refer to the VM/370: Operator's Guide for a detailed description of spool classes.

spoolid — is the spoolid number of the file that is to be changed. Each spool file has a unique spoolid.

ALL — changes all your spool files.

CLASS c2 — changes the spool class of the file to c2.

COPY nn — specifies the number of copies of the file to be spooled to the virtual output device. This option is valid for printer and punch files only. The value of nn (number of copies) must be a number from 1 through 99. For nn less than 10, the leading zero is optional.

HOLD — prevents the file from being printed, punched, or read until it is released. The file is released when the CHANGE command is issued with the NOHOLD operand specified.

NOHOLD releases the specified file from user HOLD status.

DIST distcode
changes the distribution code specified in the VM/370 directory to the distcode specified on the command line, for the specified file only. The distribution code appears on the output separators of the printer and punch output; it has no effect on reader files.

NAME fn [ft]
assigns identification to the spool file in the CMS format filename and filetype. The field, fn, is a one- to eight-character alphameric filename assigned to the file for identification. The field, ft, is a one- to eight-character alphameric filetype assigned to the file for identification. If ft is not specified, the filetype is set to blanks.

NAME dsname
assigns identification to the spool file in a non-CMS format. The field, dsname, is a 1- to 24-character field suitable for specifying OS or DOS files [for example, SYS1.SYSLIB.MYMAC].

Response

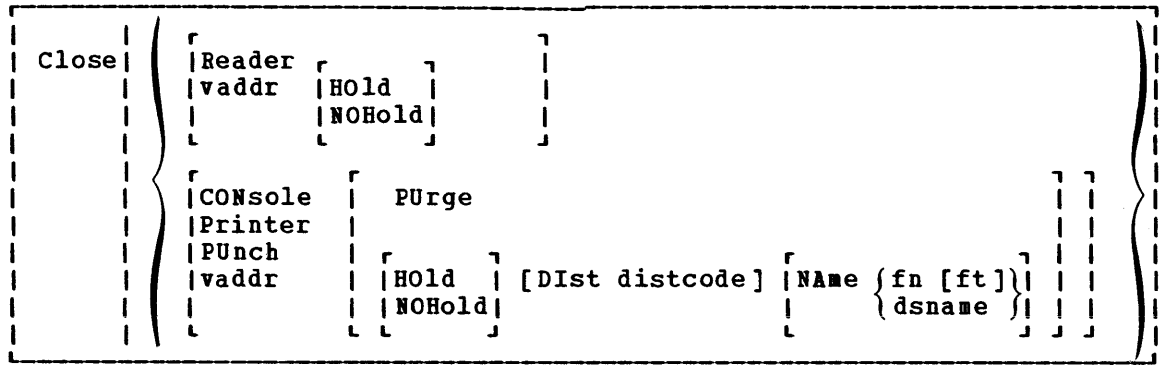
{ nnnn } FILES CHANGED
{ NO }

This is the response when you issue the CHANGE command. This is an indication of the number of files changed. It does not reflect individual alterations to a given file. This message does not appear if you have issued the CP SET IMSG OFF command line.

CLOSE

Privilege Class: G

Use the CLOSE command to terminate the spooling activity on any virtual spooled record or console device. If the file is an input reader file, the file being processed is purged unless SPOOL READER HOLD was previously specified (see the SPOOL command). The effect of HOLD or NOHOLD for a particular file established by the SPOOL command can be overridden by specifying NOHOLD or HOLD, respectively, in the CLOSE command. If the file is an output file on a printer, punch, or console, the file is either queued for output on a real unit record device, or, if the virtual output device is transferred (by use of the "SPOOL vaddr TO userid" command), the file is queued for input to the receiving user. You can specify a filename and filetype and an optional distribution code to aid in later identification of the file and its contents. The format of the CLOSE command is:



where:

READER
RDR — closes all reader spool files.

CONSOLE — closes your virtual machine's console spool file. Once a virtual console spool file is closed, it becomes a printer spool file and can be manipulated in the same way as any printer spool file (for example, it can be purged or changed).

PRINTER
PRT — closes all printer spool files.

PUNCH
PCH — closes all punch spool files.

vaddr — is the virtual address (cuu) of the device to be closed. The address may represent a reader, console, printer, or punch.

HOLD — makes the spool file being closed unavailable for further processing, until it is specifically requested or changed. This option, specified in the CLOSE command, overrides any previously specified HOLD or NOHOLD option for the files being closed.

NOHOLD — makes the spool file being closed available for further processing. Specify NOHOLD if a HOLD established by the SPOOL

command is still in effect and the current active file is not to be held.

You can release one of your own output files in HOLD status by using the CHANGE command. If an output file is spooled for another user (SPOOL FOR userid), only the receiving virtual machine user can change the file status. The originator of the file may reclaim the file by using the TRANSFER command with the FROM option. If an output file is spooled to a user as an input file (SPOOL TO userid), the HOLD option places the input file in HOLD status. The file then cannot be read by the virtual machine until it is changed to NOHOLD by the receiving virtual machine user.

If an input file is closed with the HOLD option, the file is saved and not purged from the system. The saved file is available for virtual machine and user processing and is not placed in a user hold status. Input spool files that are closed are normally purged from the virtual machine.

PURGE closes and immediately purges from the virtual machine the output spool files. No output file is produced.

DIST distcode
uses the one- to eight-character alphameric identification (distcode) on the output separators of printer and punch instead of the identification specified in the VM/370 directory. The distribution code is changed for this file only and does not affect other files or change the VM/370 directory. If the file is transferred to another user, this option has no effect.

NAME fn [ft]
assigns identification to the spool file in the CMS format filename and filetype. The field, fn, is a one- to eight-character alphameric filename assigned to the file for identification. The field, ft, is a one- to eight-character alphameric filetype assigned to the file for identification. If ft is not specified, the filetype is set to blanks.

NAME dsname
assigns identification to the spool file in a non-CMS format. The dsname field is a 1- to 24-character field suitable for specifying OS or DOS files (for example, SYS1.SYSLIB.MYMAC). Only 20 characters of the 24-character dsname are displayed by QUERY, even though a name of up to 24 characters is valid.

The table that follows shows the result of the CLOSE command, depending on how you had previously specified the SPOOL command options HOLD, NOHOLD, CONT, and NOCONT. The CLOSE command can result in a file being held, saved, or purged.

Virtual Device Status	SPOOL Command Options Set for a Virtual Device			
(CLOSE Command Setting)	NOHOLD NOCONT	HOLD NOCONT	NOHOLD CONT	HOLD CONT
Normal EOF (default CLOSE)	File released for processing	File saved	File released for processing	File held
CLOSE	File released for processing	File saved	File released for processing	File held
CLOSE HOLD	File saved	File saved	File held	File held
CLOSE NOHOLD	File released for processing	File released for processing	File released for processing	File released for processing

Note, "saved" means that the file is not purged and does not have HOLD status. A subsequent READ could read this file.

Responses

$$\left\{\begin{matrix}\text{PRT}\\\text{PUN}\\\text{CON}\end{matrix}\right\}\ \text{FILE spoolid}\ \left\{\begin{matrix}\text{TO}\\\text{FOR}\end{matrix}\right\}\ \text{userid COPY nn}\ \left\{\begin{matrix}\text{HOLD}\\\text{NOHOLD}\end{matrix}\right\}$$

This response is received if multiple copies of the file are being processed, if the file is being transferred to another user, or if the file is placed in a USER HOLD status.

COUPLE

Privilege Class: G

Use the COUPLE command to connect your virtual (non-dedicated) channel-to-channel adapter to another user's virtual channel-to-channel adapter (or to another one of your own virtual channel-to-channel adapters). The format of the COUPLE command is:

```
| COUPLE | vaddr1 [To] userid vaddr2 |
```

where:

vaddr1 is the virtual address (cuu) of your channel-to-channel adapter.

[TO] userid
is the user identification of the virtual machine to which vaddr1 is to be connected. If vaddr1 is to be connected to your own virtual machine, userid may be specified as an asterisk (*). The user must be logged on and have a virtual channel-to-channel adapter defined. If the keyword TO is omitted, the userid cannot be "T" or "TO".

vaddr2 is the virtual address (cuu) of the channel-to-channel adapter to be connected to vaddr1.

Responses

CTCA vaddr1 COUPLE TO userid vaddr2

This is the response when you issue the COUPLE command.

vaddr1 is the address of your channel-to-channel adapter.

userid is the identification of the receiving virtual machine.

vaddr2 is the address of the channel-to-channel adapter of the receiving user (or a different channel-to-channel adapter in your own virtual machine).

CTCA vaddr2 COUPLE BY userid vaddr1

This is the response sent to the user specified by userid in the COUPLE command.

vaddr1 is the address of the issuing user's channel-to-channel adapter.

userid is the identification of the user who issued the COUPLE command.

vaddr2 is the address of the channel-to-channel adapter of the receiving user.

CTCA vaddr1 DROP FROM userid vaddr

This response to the issuing user indicates that the virtual CTCA vaddr1 was already coupled when the COUPLE command was issued. The previous connection is completed. This response is always followed by the response:

CTCA vaddr1 COUPLE TO userid vaddr2

CP

Privilege Class: Any

Use the CP command if you are a CMS user and do not want to keep aware of which command environment you are in.

The CP command is treated as a "null" by the control program and therefore can precede any other command if one or more blanks separate CP from the other command. The CP command is useful because it lets the CMS user enter commands without knowing which environment (CP or virtual machine) he is in. The format of the CP command is:

```
┌──────────────────────────────────────────────────────────────┐
│ CP        │ [command]                                        │
└──────────────────────────────────────────────────────────────┘
```

where:

command is any CP command or string of CP commands that are separated by the logical line end symbol (#) and are valid for your privilege class.

Example:

```
CP QUERY FILES

   -- or --

QUERY FILES
```

can be entered from the CP or CMS command environment.

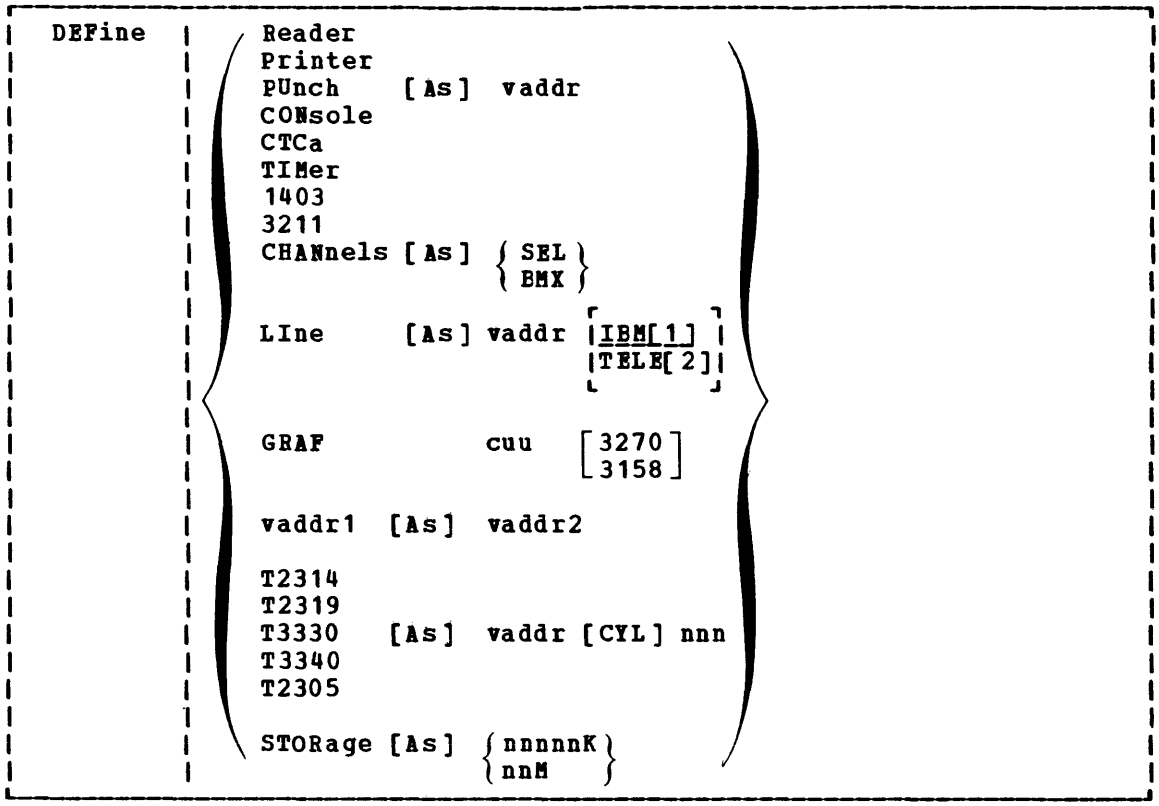
Responses

None

DEFINE

Privilege Class: G

Use the DEFINE command to alter your virtual machine configuration or channel operating mode. You can expand your configuration without making permanent changes, because the definitions are in effect for the current terminal session only. If you redefine storage or define the channel operating mode, the virtual machine is reset and IPL must be performed again. The format of the DEFINE command is:



where:

READER [AS] vaddr
RDR
adds a spooling card reader with the address specified by vaddr to the virtual machine configuration.

PRINTER [AS] vaddr
PRT
adds a spooling printer with the address specified by vaddr to the virtual machine configuration.

PUNCH [AS] vaddr
PCH
adds a spooling card punch with the address specified by vaddr to the virtual machine configuration.

CONSOLE [AS] vaddr
adds a virtual system console to the virtual machine at the address specified by vaddr.

CTCA [AS] vaddr	adds a virtual channel-to-channel adapter with the address specified by vaddr to the virtual machine configuration. The control unit address must end in zero, and must not already be in use. Once the control unit is defined, other virtual devices may not be defined for the same CTCA.
TIMER [AS] vaddr	adds a pseudo timing device with the address specified by vaddr to the virtual machine configuration.
1403 [AS] vaddr	adds a spooling 1403 printer with the address specified by vaddr to the virtual machine configuration.
3211 [AS] vaddr	adds a spooling 3211 printer with the address specified by vaddr to the virtual machine configuration. The virtual 3211 printer supports LOADVFCB and the Index feature.
CHANNELS [AS] {SEL / BMX}	redefines the channel mode of operation for the virtual machine to either selector or block multiplexer. Use of the SEL (selector channel) or BMX (block multiplexer channel) operand sets the mode of operation for all channels except virtual channel 0. Channel 0 always operates in byte-multiplexer mode. The real or virtual channel-to-channel adapter always operates in selector mode. Block multiplexer mode may enhance the virtual machine's operating system by allowing the overlap of SIO operations. This is done by reflecting a channel condition code of 0 back to the virtual machine rather than a channel busy signal. Note: The virtual machine is immediately reset when this set of operands is executed.
LINE [AS] vaddr [IBM1 / TELE2]	adds a virtual 2701/2702/2703 communication line with the address specified by vaddr to the virtual machine configuration. IBM1 indicates that an IBM-type terminal (2741, 1050, or equivalent) is on the 2701/2702/2703 line. TELE2 indicates that a teletypewriter is on the 2701/2702/2703 line.
GRAF cuu [3270 / 3158]	defines a temporary virtual 3270 or 3158 (the system console used on the System/370 Model 158) for the virtual machine. The cuu is the hexadecimal address for the device.
vaddr1 [AS] vaddr2	redefines the device represented by vaddr1 as vaddr2. The virtual address, vaddr1, must represent a defined device in the virtual machine configuration.

T2314 [AS] vaddr [CYL] nnn
T2319
T3330
T3340
T2305

adds a temporary virtual disk of the specified type to the virtual machine configuration. The vaddr specifies the address of the disk, and must not be on a virtual control unit already defined as a CTCA. CYL nn specifies the number of cylinders that the disk contains.

STORAGE [AS] { nnnnnnK / nnM }

redefines the size of the virtual storage for the virtual machine as nnnnnnK (where K represents 1024 bytes) or nnM (where M represents 1,048,576 bytes). The value specified becomes the new virtual storage size. Sizes must be in 4K increments and are limited by the maximum value in the VM/370 directory entry. The minimum size you can specify is 8K. All entries not specified in a 4K increment are rounded up to the next 4K boundary. Changing the virtual storage size (increasing or decreasing) causes a virtual system reset and clears all virtual storage to binary zeros.

Responses

Responses are generated to confirm that the desired configuration change has taken place. These responses do not appear on your terminal if you have issued the CP SET IMSG OFF command line.

type vaddr DEFINED

where the possible values for type have the following meanings:

Type	Meaning
DASD	Direct access storage device
TAPE	Magnetic tape
LINE	Communication line
RDR	Card reader
PRT	Line printer
PUN	Card punch
GRAF	Graphics device
CONS	Console
CTCA	Channel-to-channel adapter

CHANNELS = { SEL / BMX }

is the channel mode of operation for the virtual machine. This response applies to all channels except channel 0 (always a byte-multiplexer channel) and any channel that has a virtual or real channel-to-channel adapter (always a selector channel).

STORAGE = nnnnnK

The minimum storage you may specify is 8K.

DETACH

Privilege Class: G

Use the DETACH command to remove a virtual device from the virtual machine. You can detach a previously attached device even if the device is currently in use. You can also detach devices that were attached because of VM/370 directory entries or during CP system initialization. You cannot detach volumes in the system SYSOWN list or devices containing minidisks that are in use. When you detach a virtual device, it becomes inaccessible to your virtual machine. If the device was previously attached to your virtual machine by an ATTACH command, it is released and becomes available for attachment to your virtual machine, another user, or the CP system. Tape devices are automatically rewound and unloaded when detached. If you detach a device that was previously attached to your virtual machine by the operator, a message is sent to the operator informing him that the device is free. The format of the DETACH command is:

where:

vaddr is the virtual address (cuu) of the device to be detached from your virtual machine.

Responses

Messages are sent to the user, the operator who issued the command, and the primary system operator (if different from the operator who issued the command), notifying them that the DETACH was successful.

type vaddr DETACHED

This is the response you receive when you detach one of your own devices. You do not receive this response if you have issued the CP SET IMSG OFF command.

type vaddr DETACHED BY operator

This is the response you receive if an operator detaches one of your devices.

type raddr DETACHED [userid|SYSTEM]

This is the response sent to the operator if you detach a previously attached device or if the operator detaches it from a user or the system.

```
type raddr DETACHED  ┌      ┐
                     |userid| BY operator
                     |SYSTEM|
                     └      ┘
```

This is the response sent to the primary system operator if he did not issue the DETACH command and the device had been previously attached.

In the above responses, type is one of the following:

Type	Meaning
DASD	Direct access storage device
TAPE	Magnetic tape
LINE	Communication line
RDR	Card reader
PRT	Line printer
PUN	Card punch
GRAF	Graphics device
CONS	Console
CTCA	Channel-to-channel adapter
DEV	Any other device

CTCA vaddr DROP FROM userid vaddr

This is the response if the device detached was a virtual CTCA connected (via the COUPLE command) to another CTCA on the virtual machine specified by the userid. This response is always followed by the response:

```
CTCA vaddr DETACHED
```

unless the SET IMSG OFF command was issued.

DIAL

Privilege Class: Any

Use the DIAL command to logically connect a switched line, leased line, or locally attached terminal to a previously logged on multiple-access virtual machine. Once the connection is made, your terminal operates entirely under the control of that virtual machine. The DIAL command matches your terminal to the equivalent type defined in the multiple-access virtual machine. If no matching terminal type exists, the connection cannot be made and an error message is issued. The format of the DIAL command is:

```
| DIAL | userid [vaddr] |
```

where:

userid — is the identification of a virtual machine that is currently logged on.

vaddr — is the address of the virtual communication line to which the connection is made.

Notes:

1. A DIAL command is accepted only at logon time, and only as a substitute for a LOGON command. The type of terminal used must be supported by both VM/370 and the multiple-access virtual machine. The only exception is the 3277 Display Station Model 1 (480-character screen). This display terminal, if used, must be supported by the multiple-access virtual machine. VM/370 support of this model is limited to the DIAL facility only. See the VM/370: Terminal User's Guide for details on running and gaining access to multiple-access machines.

2. If the DIAL command is issued from a real 3277 terminal, the virtual system must use the CP command RESET to drop the dialed connection. DIAL is not supported for the 3066 system console.

3. The CP DIAL command is not supported for terminals that are using NCP lines in a 3704/3705 control unit.

Responses

DIALED TO userid vaddr

is the message sent to the user indicating that a logical connection has been made.

{GRAF raddr / LINE raddr / DEV rid} DIALED TO userid DIALED = nnn

is the response to the primary system operator. It indicates a successful connection to the virtual machine (userid) and the total number of VM/370 lines (nnn) currently connected to other virtual machines. DEV rid indicates the resource identification of a 3704/3705 line.

Note: The terminal remains connected to and under the control of the virtual machine until that virtual machine terminates the

communication. At that time the user receives the following message:

```
DROP FROM userid vaddr
```

this message is sent to the user when the line is disabled.

```
{GRAF raddr}
{LINE raddr} DROP FROM userid DIALED = nnn
{DEV  rid  }
```

is the message sent to the primary system operator.

DISCONN

Privilege Class: Any

Use the DISCONN command to disconnect the terminal from the VM/370 system: the virtual machine continues operation. When DISCONN is issued from the virtual machine, a disconnect-time message is printed at the virtual machine's terminal and at the primary system operator's console. The terminal remains disconnected until it is reconnected via a LOGON command. The virtual machine is logged off 15 minutes after an attempt is made to read from the terminal or if the virtual machine goes into a disabled WAIT state.

If your terminal connection is broken because of terminal, line, or TP control unit errors, CP places the virtual machine in disconnect mode for up to 15 minutes and your virtual machine does not continue to run. If you log on within 15 minutes, your virtual machine can continue operating. If you do not log on within the 15-minute interval, the virtual machine is logged off.

Unless the CP command, SPOOL CONSOLE START, is issued to spool the virtual console output, all "writes" or output messages to the virtual console are ignored. When the terminal is reconnected via the normal LOGON procedure, the terminal is placed in CP console function mode. To resume execution of the virtual machine, enter the BEGIN command. The format of the DISCONN command is:

```
DISConn | [HOld]
```

where:

HOLD specifies that the communication line is not to be disabled. This option allows you to disconnect your terminal, and, at the same time, to avoid the process of telephone dialing into the system to access your virtual machine again. If specified, control returns to CP and the "VM/370 online" message is displayed.

Responses

When the DISCONN command is issued, the disconnect time message is issued.

```
DISCONNECT AT hh:mm:ss zone weekday mm/dd/yy
```

is the response to the user who issued the command.

```
{GRAF raddr}
{LINE raddr} DISCONNECT userid USERS = nnn
{DEV  rid  }
```

is the response to the primary system operator informing him that the user represented by "userid" has been disconnected from the VM/370 system. The "nnn" is the total number of users remaining in the system. The response "rid" indicates the resource identification.

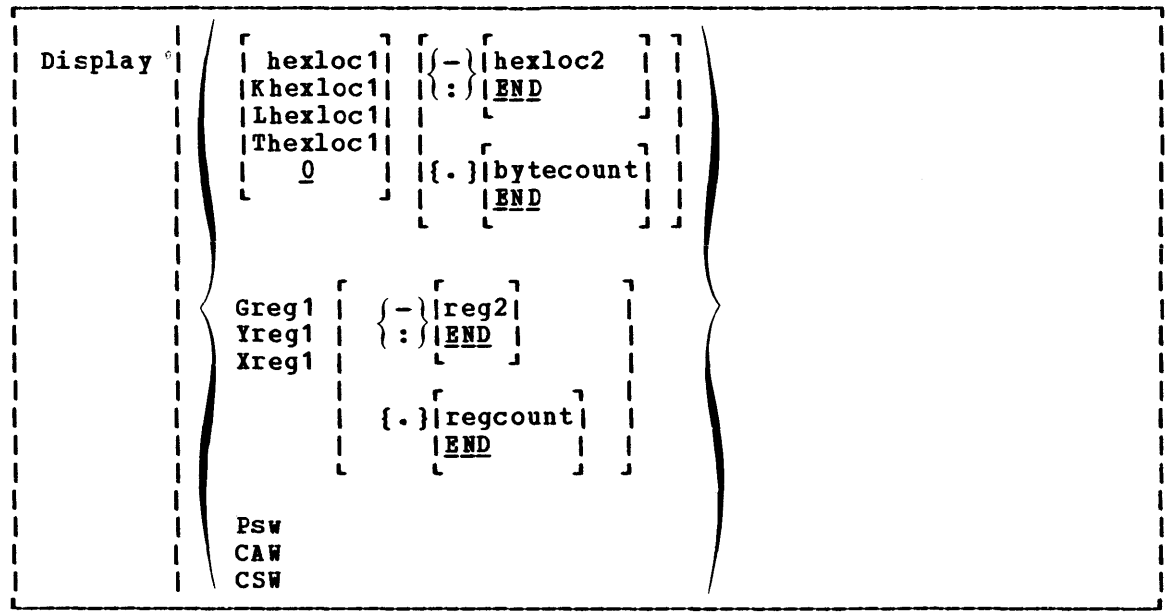
DISPLAY

Privilege Class: G

Use the DISPLAY command to examine the following virtual machine components:

- Virtual storage locations
- General registers
- Floating-point registers
- Control registers
- Program status word (PSW)
- Channel address word (CAW)
- Channel status word (CSW)

If a command line with an invalid operand is entered, the DISPLAY command terminates when it encounters the invalid operand; however, any previous valid operands are processed before termination occurs. Storage locations, registers, and control words can be displayed using a single command line. The format of the DISPLAY command is:



where:

hexloc1
Lhexloc1
Thexloc1
Khexloc1
0

is the first, or only, hexadecimal storage location whose contents are to be displayed at the terminal. If L is specified, the storage contents are displayed in hexadecimal. If T is specified, the storage contents are displayed in hexadecimal, with EBCDIC translation. If K is specified, the storage keys are displayed in hexadecimal.

If hexloc1 is followed by a period and is not on a fullword boundary, it is rounded down to the next lowest fullword.

If hexloc1 is not specified, the display begins at storage location 0. If L, T, or K are entered either

without any operands, or followed immediately by a blank, the contents of all storage locations are displayed. If L, T, or K are not specified and this is the first operand, then the default value of zero is assumed. The address, hexloc1, may be one to six hexadecimal digits; leading zeros are optional.

{-}hexloc2
{:}END

is the last of the range of hexadecimal storage locations whose contents are to be displayed at the terminal. Either - or : must be specified to display the contents of more than one location by storage address. If hexloc2 is not specified, the contents of all storage locations from hexloc1 to the end of virtual storage are displayed. If specified, hexloc2 must be equal to or greater than hexloc1 and within the virtual storage size. The address, hexloc2, may be from one to six hexadecimal digits; leading zeros are optional.

{.}bytecount
END

is a hexadecimal integer designating the number of bytes of storage (starting with the byte at hexloc1) to be displayed at the terminal. The period, ., must be specified to display the contents of more than one storage location by byte count. The sum of hexloc1 and bytecount must be an address that does not exceed the virtual machine size. If this address is not on a fullword boundary, it is rounded up to the next highest fullword. The value, bytecount, must have a value of at least one and may be from one to six hexadecimal digits; leading zeros are optional.

Greg1

is a decimal number from 0-15 or a hexadecimal integer from 0-F representing the first, or only, general register whose contents are to be displayed at the terminal. If G is specified without a register number, the contents of all the general registers are displayed at the terminal.

Yreg1

is an integer (0, 2, 4, or 6) representing the first, or only, floating-point register whose contents are to be displayed at the terminal. If Y is specified without a register number, the contents of all of the floating-point registers are displayed at the terminal.

Xreg1

is a decimal number from 0-15 or a hexadecimal number from 0-F representing the first, or only, control register whose contents are to be displayed at the terminal. If X is specified without a register number, the contents of all of the control registers are displayed at the terminal. If Xreg1 is specified for a virtual machine without extended mode operations available, only control register 0 is displayed.

{-}reg2
{:}END

is a number representing the last register whose contents are to be displayed at the terminal. Either - or : must be specified to display the contents of more than one register by register number. If reg2 is not specified, the contents of all registers from reg1 through the last register of this type are displayed.

The operand, reg2, must be equal to or greater than reg1. If Greg1 or Xreg1 are specified, reg2 may be a decimal number from 0-15 or a hexadecimal number from 0-F. If Yreg1 is specified, reg2 may be 0, 2, 4, or 6. The contents of registers reg1 through reg2 are displayed at the terminal.

{.}regcount
END
: is a decimal number from 1 to 16 or a hexadecimal number from 1 to F specifying the number of registers (starting with reg1) whose contents are to be displayed at the terminal. If the display type G or X is specified, regcount can be a decimal number from 1 to 16 or a hexadecimal number from 1 to F. If display type Y is specified, regcount must be 1, 2, 3, or 4. The sum of reg1 and regcount must be a number that does not exceed the maximum register number for the type of registers being displayed.

PSW
: displays the current virtual machine PSW (program status word) as two hexadecimal words.

CAW
: displays as one hexadecimal word the contents of hexadecimal location 48 (channel address word).

CSW
: displays as two hexadecimal words the contents of the channel status word (doubleword at hexadecimal location 40).

When multiple operands are entered on a line for location or register displays, the default display type is the same as the previous explicit display type. The explicit specification of a display type defines the default for subsequent operands for the current display function. Blanks are used to separate operands or sets of operands if more than one operand is entered on the same command line. Blanks must not be used to the right or left of range or length delimiters (: - .), unless it is intended to take the default value of the missing operand defined by the blank. For example:

```
display 10 20 T40 80 G12 5 L60-100
```

displays the following:

```
hexadecimal location 10
hexadecimal location 20
hexadecimal location 40 with EBCDIC translation
hexadecimal location 80 with EBCDIC translation
general register 12
general register 5
hexadecimal locations 60 through 100
```

Responses

One or more of the following responses is displayed, depending upon the operands specified.

Locations

xxxxxx word1 word2 word3 word4 [key] *EBCDIC TRANSLATION*

This is the response you receive when you display storage locations; xxxxxx is the hexadecimal storage location of word1. Word1 is displayed (word-aligned) for a single location specification. Up to four words are displayed on a line, followed, optionally, by an EBCDIC translation of those four words. Periods are printed for unprintable characters. Multiple line are used (if required) for a range of locations. If translation to EBCDIC is requested (Thexloc), alignment is made to the next lower 16-byte boundary; otherwise, alignment is made to the next lower fullword boundary. If the location is at a 2K page boundary, the key for that page is also displayed.

Keys:

xxxxxx TO xxxxxx KEY = kk

This is the response you receive when you display storage keys; xxxxxx is a storage location and kk is the associated storage key.

General Registers

GPR n = genreg1 genreg2 genreg3 genreg4

This is the response you receive when you display general registers; n is the register whose contents are genreg1. The contents of the following consecutive registers are genreg2 and so on. The contents of the registers are displayed in hecadecimal. Up to four registers per line are displayed for a range of registers. Multiple lines are displayed if required, with a maximum of four lines needed to display all 16 general registers.

Floating-Point Registers

FPR n = xxxxxxxxxxxxxxxx .xxxxxxxxxxxxxxxx E xx

This is the response you receive when you display floating-point registers; n is the even-number floating-point register whose contents are displayed on this line. The contents of the requested floating-point registers are displayed in both the internal hexadecimal format and the E format. One register is displayed per line. Multiple lines are displayed for a range of registers.

Control Registers

ECR n = ctlreg1 ctlreg2 ctlreg3 ctlreg4

This is the response you receive when you display control registers; n is the register whose contents are ctlreg1. The contents of the following consecutive registers are ctlreg2 and so on. The contents of the requested control registers are displayed in hexadecimal. Up to four registers per line are displayed. Multiple lines are displayed if required.

PSW

PSW = xxxxxxxx xxxxxxxx

The contents of the PSW are displayed in hexadecimal.

CAW

CAW = xxxxxxxx

The contents of the CAW (hexadecimal location 48) are displayed in hexadecimal.

CSW

CSW = xxxxxxxx xxxxxxxx

The contents of the CSW (hexadecimal location 40) are displayed in hexadecimal.

Press the Attention key (or its equivalent) to terminate this function while data is being displayed at the terminal. When the display terminates, another command may be entered.

DUMP

Privilege Class: G

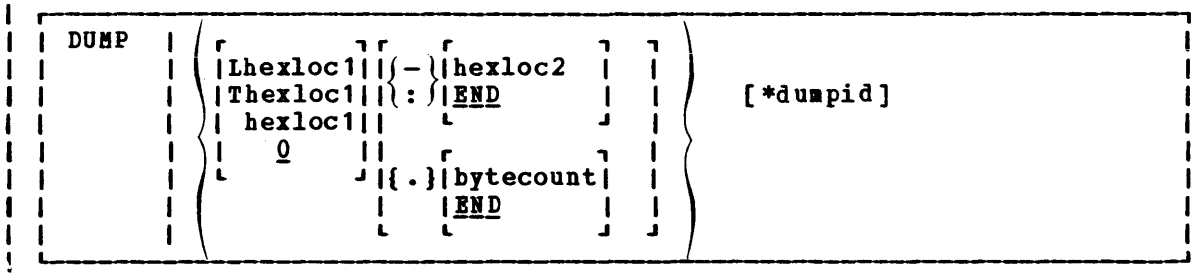
Use the DUMP command to print the contents of various components of the virtual machine on the virtual spooled printer. The following items are printed:

- Virtual program status word (PSW)
- General registers
- Floating-point registers
- Control registers (if you have the ECMODE option specified in your VM/370 directory entry)
- Storage keys
- Virtual storage locations

The DUMP command prints the virtual PSW and the virtual registers (general, floating-point, and control). If only this information is desired, at least one virtual address must be specified, such as:

```
DUMP 0
```

The output format for the virtual storage locations is eight words per line with EBCDIC translation on the right. Each fullword consists of eight hexadecimal characters. All the rest of the information (PSW, general floating-point and storage keys) is printed in hexadecimal. If you have the ECMODE option in your VM/370 directory entry, the control registers are also printed. To print the dump on the real printer, a CLOSE command must be issued for the spooled virtual printer. The format of the DUMP command is:



where:

Lhexloc1
Thexloc1
hexloc1
0
is the first or only hexadecimal storage location to be dumped. If you enter L or T without operands, the contents of all virtual storage locations are dumped.

The address, hexloc1, may be one to six hexadecimal digits; leading zeros are optional. If hexloc1 is not specified, the dump begins at storage location 0.

If hexloc1 is followed by a period and is not on a fullword boundary, it is rounded down to the next lowest fullword.

{-}hexloc2
{:}END — is the last hexadecimal storage location whose contents are to be dumped to the printer. The operand, hexloc2, must be equal to or greater than hexloc1 and within the virtual storage size. To dump to the end of storage, you can specify END instead of hexloc2 or you can leave the field blank, since the default is END. If you specify :END or -END, the contents of storage from hexloc1 to END are dumped. The contents of storage locations hexloc1 through hexloc2 are printed with EBCDIC translation at the printer. The operand, hexloc2, may be from one to six hexadecimal digits; leading zeros are optional.

{.}bytecount
END — is a hexadecimal integer designating the number of bytes of storage (starting with the byte at hexloc1) to be dumped to the printer. The period, ., must be specified to dump the contents of more than one storage location by byte count. The sum of hexloc1 and bytecount must be an address that does not exceed the virtual machine size. If this address is not on a fullword boundary, it is rounded up to the next highest fullword. The value, bytecount, must be one or greater and can be no longer than six hexadecimal digits. Leading zeros are optional.

*dumpid — can be entered for descriptive purposes. If specified, it becomes the first line printed preceding the dump data. Up to 100 characters, with or without blanks, may be specified after the asterisk prefix. No error messages are issued, but only 100 characters are used, including asterisks and embedded blanks.

Usage:

Normally, you should define beginning and ending dump locations in the following manner:

```
dump Lhexloc1-hexloc2
dump Lhexloc1.bytecount
dump Lhexloc1-hexloc2 hexloc1.bytecount * dumpid
```

If, however, a blank follows the type character (L or T) or the character and the hexloc, the default dump starting and ending locations are assumed to be the beginning and/or end of virtual storage. Blanks are used to separate operands or sets of operands if more than one operand is entered on the same command line. Blanks must not be used to the right or left of range or length delimiters (: - .), unless it is intended to take the default value of the missing operand defined by the blank. Thus, all of the following produce full storage dumps:

```
dump l      dump t:       dump 0-end
dump t      dump 1.       dump 1:end
dump -      dump t.       dump t:end
dump :      dump 0-       dump 0:end
dump .      dump 0:       dump 1.end
dump 1-     dump 0.       dump t.end
dump t-     dump 1-end    dump 0.end
dump 1:     dump t-end
```

The following produces three full dumps:

```
dump 1 . t
dump - . :
```

Responses

DUMPING LOC hexloc

As the dump is processing, the following message is displayed at the terminal indicating that the dump is continuing from the next 64K boundary: where hexloc is the segment (64K) boundary address for the dump continuation, such as 020000, 030000, or 040000.

If you press the Attention key, or its equivalent, on the terminal while the message is being displayed, the dump function is terminated.

COMMAND COMPLETE

This response indicates normal completion of the dump function.

ECHO

Privilege Class: G

Use the ECHO command to place the terminal in the ECHO environment. When in the ECHO environment, any input line entered is transmitted unchanged back to the terminal a specified number of times. To terminate this transmission (for example, when you want to enter a different data line), press the Attention key (or its equivalent). When the specified number of lines is displayed or the Attention key is pressed, another read to the terminal is issued to accept another data line. Note that no line editing is done; thus, the output line is the same as the input line and may contain any of the logical line editing symbols. The format of the ECHO command is:

where:

nn is the number of times the line is to be sent. The default is 1. An invalid entry (that is, one that is greater than 99 or contains non-numeric characters) is treated as 1.

Responses

ECHO ENTERED; TO TERMINATE TEST, TYPE END

This message is displayed after the ECHO command is invoked to indicate that the ECHO environment has been entered.

ENTER LINE

This message requests the input line to be entered. A reply of END returns the terminal to the CP command environment.

EXTERNAL

Privilege Class: G

Use the EXTERNAL command to simulate an external interrupt to the virtual machine and return control to that machine. This simulates pressing the interrupt key on the real computer console, or other functions which cause an external interrupt. Control is given to the virtual machine immediately. The format of the EXTERNAL command is:

where:

code is the interrupt code, a hexadecimal number to be associated with the external interrupt. Valid codes are 1005 (CPU Timer), 1004 (Clock Comparator), and all codes less than or equal to X'FF'. The default is the External Interrupt Button on the system console, X'40'.

Responses

None. Since control is given to the virtual machine, any response is from virtual machine processing.

INDICATE

Privilege Class: G

Use the INDICATE command to display at your terminal, the use of and contention for major system resources. Use INDICATE LOAD to display system load conditions. Use INDICATE USER to display the total amount of certain resources used by your virtual machine during the current terminal session. Use the INDICATE USER command before and after the execution of a program to indicate the execution characteristics of that program in terms of the resources used.

The format of the INDICATE command is:

where:

LOAD displays CPU use, CPU contention, main storage use, and main storage contention.

USER * displays the amounts of system resources used by your virtual machine in the current terminal session.

Response for INDICATE LOAD

CPU-nnn% Q1-nn Q2-nn STORAGE-nnn% RATIO-n.n

where:

- CPU-nnn% - is the percentage of total CPU usage. The CPU figure is a smoothed value of the percentage of time that the system is running. The value is smoothed because instantaneous values can be misleading.
- Q1-nn Q2-nn - represent the contention for the CPU in terms of the smoothed number of users in queue 1 and queue 2 (maintained by the scheduler).
- STORAGE-nnn% - is the percentage of main storage usage. This is a smoothed ratio of the sum of the estimated working sets of users in queue 1 and queue 2 to the number of pageable pages in the system.
- RATIO-n.n - represents the contention for main storage. This scheduler contention ratio is a smoothed value and is defined as:

$$RATIO = \frac{E+M}{M}$$

where:

E the number of users waiting to be allocated main storage by the scheduler and, therefore, temporarily resident in the scheduler's eligible lists.

M the number of users in queue 1 and queue 2.

Thus, RATIO is the ratio of users now active to users being serviced and is 1.0 for optimum response. Optimum response occurs when enough storage exists to accommodate all active users, assuming that the system at this time is not CPU bound.

If E and M are both zero, the value of RATIO is set to 1.0.

When RATIO=1.5 and M=8, then 4 users are in the eligible lists waiting to be allocated main storage space by the scheduler. While in the eligible list, the users are subject to scheduler discrimination, as defined by the biased scheduler.

Response for INDICATE USER

```
PAGES: RES-nnnn  WS-nnnn READS=nnnnnn WRITES=nnnnnn  DISK-nnnn DRUM-nnnn
VTIME=nnn:nn TTIME=nnn:nn SIO=nnnnnn RDR-nnnnnn PRT-nnnnnn PCH-nnnnnn
```

where:

- RES-nnnn - is the current number of your virtual storage pages resident in main storage. This number is taken at an instant of time during the execution of the INDICATE command.
- WS-nnnn - is the most recent system estimate of your working set size.
- READS=nnnnnn - is the total number of page reads that have occurred for you since you logged on or since the last ACNT command was issued for your virtual machine.
- WRITES=nnnnnn - is the total number of pages written for you since you have logged on or since the last ACNT command was issued for your virtual machine.
- DISK-nnnn - is the current number of virtual pages allocated for you on the system paging disk. This number is taken at an instant of time during the execution of the INDICATE command.
- DRUM-nnnn - is the current number of virtual pages allocated for you on the system paging drum. This number is taken at an instant of time during the execution of the INDICATE command.
- VTIME=nnn:nn - is your total virtual machine time since you logged on or since the last ACNT command was issued for your virtual machine.
- TTIME=nnn:nn - is your total virtual machine time and total CPU time (virtual and overhead) that you have used since you logged on or since the last ACNT command was issued for your virtual machine.
- SIO=nnnnnn - is the total number of non-spooled I/O requests that you have issued since you logged on or since the last ACNT command was

INDICATE

issued for your virtual machine.

- RDR-nnnnnn - is the total number of virtual cards read since you logged on or since the last ACNT command was issued for your virtual machine.

- PRT-nnnnnn - is the total number of virtual lines printed since you logged on or since the last ACNT command was issued for your virtual machine.

- PCH-nnnnnn - is the total number of virtual cards punched since you logged on or since the last ACNT command was issued for your virtual machine.

IPL

Privilege Class: G

Use the IPL command to simulate an initial program load function for a virtual machine. IPL simulates the LOAD button and the device address switches on the real computer console. The specified virtual address is accessed and the required input/output operations are performed to retrieve the IPL data. Optionally, the IPL procedure can be stopped just before loading the virtual PSW except when initial program loading a named system. Also, parameters can be passed to the virtual machine's general registers. When the simulated load function is complete, CP initiates execution of the virtual machine by loading the IPL PSW which was stored during the simulation process. The format of the IPL command is:

```
| Ipl  | {                    ┌         ┐          }                          |
|      | {                    |CLear    |          }                          |
|      | { vaddr [cylno]      |NOCLear  | [STOP]   } [PARM {p1 p2... } ]     |
|      | {                    └         ┘          }                          |
|      | {                                         }                          |
|      | { systemname                              }                          |
```

where:

```
              ┌        ┐
vaddr [cylno] |CLEAR   | [STOP] [PARM {p1 p2...}]
              |NOCLEAR |
              └        ┘
```

simulates the IPL function when loading by device address.

The address, vaddr, is the virtual address (cuu) of the device that contains the nucleus to be loaded.

The number, cylno, is the cylinder containing the IPL data. If this operand is specified, CP loads the IPL data from the specified virtual cylinder instead of from the default, virtual cylinder zero. This operand is valid only for virtual direct storage devices.

The CLEAR operand clears the virtual storage space to binary zeros before the operating system is loaded; NOCLEAR does not clear storage. Both of these operands are invalid if the user specifies systemname in the IPL command line.

The STOP operand stops the virtual machine during the IPL procedure just before the initial PSW is loaded. The STOP operand provides the virtual simulation of the IPL procedure for a real machine in instruction step mode. The STOP operand is invalid with the IPL systemname form of the command. When the virtual machine stops, you can issue CP commands. For example, if you are loading OS or OS/VS into your virtual machine, you can use CP commands to store data into low storage to load an alternate nucleus or to alter the size of virtual storage. To restart the virtual machine, issue the BEGIN command.

The PARM p1 p2... operand passes up to 64 bytes of data (including embedded blanks) to your virtual machine's general registers (four bytes per register), starting with the high order byte of general register 0, whenever PARM is specified,

the remaining characters in the command line are treated as parameters to be passed to the virtual machine; therefcre, PARM must be the last operand entered on the command line.

systemname [PARM {p1 p2...}]
simulates the IPL function when loading a named system that was previously saved via the SAVESYS command.

The systemname operand is the name of the previously saved system. It is loaded into virtual storage and given control. For more information about saved systems, see the VM/370: System Programmer's Guide.

The PARM p1 p2... operand passes up to 64 bytes of data (including embedded blanks) to your virtual machine's general registers (4 bytes per register), starting with the high order byte of general register 0. Whenever PARM is specified, the remaining characters in the command line are treated as parameters to be passed to your virtual machine; therefore, PARM must be the last operand entered on the command line.

Note: Care must be used when passing parameters to a named system (systemname). Named systems expect certain registers to be initialized when they are given control. Indiscriminate use of the PARM option could overlay a previously initialized register causing unpredictable results.

Responses

After a successful IPL, any responses you receive are those from the operating system that was loaded and initialized.

LINK

Privilege Class: G

Use the LINK command to make a device that is associated with another virtual machine available to your virtual machine configuration, based upon information in that user's VM/370 directory entry. The format of the LINK command is:

```
| LINK | [To] userid vaddr1 [As] vaddr2 [mode] [[PASS=] password] |
```

where:

[TO] userid — is the name of the user whose VM/370 directory is to be searched for device vaddr1. An asterisk (*) is used to specify that the device is in your own VM/370 directory. If the keyword TO is omitted, the userid may not be TO or T.

vaddr1 — is the virtual device address (cuu) in the VM/370 directory for that userid.

[AS] vaddr2 — is the virtual address (cuu) which is to be assigned to the device for your virtual machine. If the keyword AS is omitted, vaddr may not be A. If your virtual machine has the ECMODE option, any address up to X'FFF' is valid; otherwise, any address up to X'5FF' is valid.

mode — is the access mode; the primary access requested (read-only, write, or multiple), and the alternate access (read-only or write) desired if the primary access is not available. Valid modes are:

Mode	Meaning
R	Read-only access. The link is not done if any other user has the disk in write status. R is the default mode if the link is to another userid.
RR	Read-only access. The link is established even if another user has the disk in write status.
W	Write access. The link is not done if any other user has the disk in read or write status.
WR	Write and read access. If another user has the disk in read or write status, an alternate access of read-only is acceptable.
M	Multiple access. This means that a write-link is to be given to the disk unless another user already has write access to it, in which case no link is to be done.
MR	Write-link. If another user already has write access to the disk, a read-link is to be done.
MW	Write-link. This link is established in all cases.

Caution: Multiple write access under CMS can produce unpredictable results.

If the mode is omitted, the default is R if the userid is another user; if you are linking to one of your own

disks, the default is the "user access mode" of either R, W, or M as specified in the VM/370 directory for your disks.

PASS= password is a one- to eight-character string that must match the access mode password for device vaddr1 in the VM/370 directory for the user (userid) specified. The password should be specified only when the LINK is executed by a virtual machine (for example, from CMS), since the password is not print suppressed when included with the LINK command. The password cannot be the same as any of the access modes (R, RR, W, WR, M, MR, or MW) if the default mode is to be used.

Note: The access mode password should not be confused with a user password.

If you link to one of your own disks, no password is required. Also, if the link is to a device whose password is ALL, meaning that the device can be used by all users, the password is not required. However, if the link is to any other userid, a password for the desired device must be provided.

Note: The access allowed by the LINK command to the vaddr1 device belonging to userid is summarized below. You read the columns down to determine the type of link that results. The first row indicates the primary (and, optionally, the alternate) access mode requested. The second row indicates whether read, write, or multiple passwords exist in the VM/370 directory for the disk being linked. The third row indicates whether the disk is already being used, and if so, the mode of its access. The last row indicates the type of link established. For example, the third column is interpreted as follows: if you request a read access link (R) to a disk that has a read password defined and that already is accessed in read mode, you can establish a read link.

Primary access requested:	R	R	R	R	R	W	W	W	W	W	W	M	M	M	M	M	M
alternate access (if any):					R				R		R					R	W
Read password in directory:	N	Y	Y	Y	Y												
Write password in directory:						N	Y	Y	Y	Y	Y						
Mult. password in directory:												N	Y	Y	Y	Y	Y
Any existing links:		N	R	W	W		N	R	R	W	W		N	R	W	W	W
Access established:	N	R	R	N	R	N	W	N	R	N	R	N	W	W	N	R	W

where: N=no or none; R=read; W=write; M=multiple; Y=yes

Responses

ENTER READ PASSWORD:
****

Type the read password over the mask to obtain read access to the desired disk.

ENTER WRITE PASSWORD:
****

Type the write password over the mask to obtain write access to the desired disk.

```
ENTER MULT PASSWORD:
■■■■■■■■■
```

Type the multiple password over the mask to obtain write access to a disk for which other users may already have access.

Note: If LINK is issued from a virtual machine with the password included on a command line, and the password is incorrect, then CP counts these incorrect passwords. If a total of ten such incorrect passwords is entered, the LINK command from a virtual machine is subsequently disallowed for that user for the remainder of the session. LINK can still be issued directly from the terminal (that is, in CP command mode), or the LINK command can be reinstated as a valid command from your virtual machine by logging off and logging on again. (This procedure is designed to protect password security if a virtual machine issues the LINK command repeatedly with trial passwords.)

```
DASD vaddr2 LINKED R/O
```

This response indicates that a read-only link to the given disk is established, for a LINK request with a mode of R or RR, and that no other users are linked to the same disk in read/write mode.

```
DASD vaddr2 LINKED R/W
```

This response indicates that a read/write link to the given disk is established, for a LINK request with a mode of W, WR, M, MR, or MW, and that no other users are linked to the same disk.

```
DASD vaddr2 LINKED R/O; R/W BY {nnn USERS}[; R/O BY {nnn USERS}]
                               {userid    }[       {userid    }]
```

This response indicates that a read-only link to the given disk is established for a LINK request with a mode of RR, but warns that the disk is in read/write use by some users and possibly in read use by some users. If only one user has access, the number of users (nnn USERS) is replaced by userid.

```
DASD vaddr2 LINKED R/W; R/O BY {nnn USERS}
                               {userid    }
```

This response indicates that a read/write link to the given disk is established for a LINK request with a mode of M, MR, or MW, and informs you that the disk is also in read-only use by userid or by nnn users. (No other users have a read/write link to the disk.)

```
DASD vaddr2 LINKED R/W; R/W BY {nn USERS}[; R/O BY {nnn USERS}]
                               {userid  }[       {userid    }]
```

This response indicates that a read/write link to the given disk is established for a LINK request with a mode of MW, but warns you that the disk is also in read/write use by some users and possibly in read use by some users. If only one user has access, the number of users (nnn USERS) is replaced by userid.

LOADVFCB

Privilege Class: G

Use the LOADVFCB command to specify the forms control for a virtual spooled 3211 printer. The format of the LOADVFCB command is:

```
LOADVFCB | vaddr   FCB  name  [Index [nn]]
```

where:

vaddr is the virtual device address (cuu) of the virtual 3211 spooled printer.

FCB is a required reserved word meaning Forms Control Buffer.

name is a system-defined name for the 3211 FCB image which is to be the controlling virtual FCB image.

There is only one VM/370 FCB image provided; its name is FCB1 and its format is as follows:

Space 6 lines/inch
Length of page 66 lines

Line Represented	Channel Skip Specification
1	1
3	2
5	3
7	4
9	5
11	6
13	7
15	8
19	10
21	11
23	12
64	9

INDEX [nn]
is the number of the print position that is the first print position. The value, nn, must be a number from 1 through 31; a leading zero need not be specified. If the keyword INDEX is specified without a value, the index defaults to the value specified in the FCB macro. See the VM/370: System Programmer's Guide for a discussion of the FCB macro and forms control images.

Note: The LOADVFCB command may be used with installations that do not have a 3211. The virtual machine's VM/370 directory entry must indicate a 3211, even though the program and operating system have a 1403 defined. Then the LOADVFCB command can be used to obtain a virtual forms control image for 1403 printers so that programs that use printer overflow sensing may be spooled to disk.

Responses

None.

LOGOFF

Privilege Class: Any

Use the LOGOFF command to terminate virtual machine execution and disconnect your virtual machine from the VM/370 system. This command causes all active spool files to be closed, temporary disks to be relinquished, dedicated devices to be detached, and an accounting record to be created for the user. The format of the LOGOFF command is:

```
| LOGoff | [HOld]                                                    |
| LCGout |                                                           |
```

where:

HOLD retains the connection for a switched communication line to enable you to logon without redialing the VM/370 system.

Usage

You should always logoff and not only turn power off on the terminal. Terminal power off is not synonymous with logoff.

If you turn power off at the terminal instead of logging off, logoff occurs by one of the following methods:

- Remote Typewriter Terminal--Logoff takes place after a 15-minute interval has elapsed. This occurs if no attempt is made to power on the terminal and re-establish communications with the still logged-on virtual machine during this 15-minute period.
- 3270 Display Terminal--Logoff only takes place 15 minutes after VM/370 discovers that the terminal has been turned off (that is, VM/370 attempts to send a message to the terminal, but gets back an error code indicating that the terminal is turned off). Because many hours may pass before VM/370 discovers that the terminal is turned off, you run the risk of compromising the security of the virtual machine and data files. Anyone turning the 3270 power back on has access to the virtual machine without logging on. This is because the machine is still logged on, although inactive.

Responses

CONNECT= hh:mm:ss VIRTCPU= mmm:ss.hs TOTCPU= mmm:ss.hs

where:

CONNECT hh:mm:ss is the actual clock time spent in the current terminal session in hours:minutes:seconds.

VIRTCPU mmm:ss.hs is the virtual CPU time used in the current terminal session in minutes:seconds.hundredths of seconds.

TOTCPU mmm:ss.hs is the total CPU time (including virtual and overhead) used in the current terminal session in minutes:seconds.hundredths of seconds.

These times are either the elapsed time for the entire terminal session or the elapsed time since the ACNT command was entered for this user.

LOGOFF AT hh:mm:ss zone weekday mm/dd/yy

is the response for a logoff.

{GRAF raddr / LINE raddr / DEV rid} LOGOFF AS userid USERS = nnn

is the normal response to the primary system operator. DEV rid specifies the resource identification.

{GRAF raddr / LINE raddr / DEV rid} LOGOFF AS userid USERS = nnn FORCED

is the response to the primary system operator if the logoff is forced by a line timeout or a terminal power-off.
DEV rid specifies the resource identification.

USER DSC LOGOFF AS userid USERS = nnn

is the response to the primary system operator when logoff occurs for a user who had previously disconnected using the DISCONN command.

LOGON

Privilege Class: Any

Use the LOGON command to identify yourself to the VM/370 system and to access that system. Upon successful logon, VM/370 creates a virtual machine configuration from information in the VM/370 directory. The LOGON command name may not be entered using any line-editing symbols, but the operands may use these symbols. See the VM/370: Terminal User's Guide for a detailed description of logon procedures. If you use LOGON because a teleprocessing line or terminal error disconnected you from your virtual machine, you have 15 minutes to logon again. If you do not log on within 15 minutes, your virtual machine automatically logs off. In this case, you may have to reconstruct files and restart jobs interrupted by the teleprocessing line or terminal error. The format of the LOGON command is:

```
| Logon | userid [password] [Mask] [Noipl]
| Login |
```

where:

userid is the identifier assigned to you in the VM/370 system.

password is your password. Specify this field if no protection (that is, masking characters) is desired.

MASK types masking characters to cover the password on typewriter terminals without the print inhibit feature. The mask types on the line following a prompting message from VM/370 requesting you to enter your password. Should you forget to ask for masking when you type LOGON, you can press the carriage return after the prompt for the password types, and VM/370 then types out the masking characters.

NOIPL specifies that the IPL device or name in the VM/370 directory should not be used for an automatic IPL.

Responses

ENTER PASSWORD:

indicates that the userid has been accepted. You should type in the password, or signal a carriage return if a mask is desired for the password, and MASK was not included on the command line.

LOGMSG- hh:mm:ss mm/dd/yy

indicates the time and date at which the system log message was generated or most recently revised. If you wish to see all of the system log messages, you must issue the CP command QUERY LOGMSG. Any lines of the log message for which the first character is an asterisk are displayed at this point.

```
FILES: {nnn} RDR, {nnn} PRT, {nnn} PUN
       {NO }      {NO }      {NO }
```

This message is omitted if all counts are zero, otherwise it indicates the number of spool files that exist for you at logon time.

```
LOGON AT hh:mm:ss zone weekday mm/dd/yy
```

-- or --

```
RECONNECTED AT hh:mm:ss zone weekday mm/dd/yy
```

indicates the time, day of the week, and date at which the LOGON or RECONNECT is complete.

```
{GRAF raddr}
{LINE raddr} LOGON AS userid     USERS = nnn
{DEV  rid  }
```

-- or --

```
{GRAF raddr}
{LINE raddr} RECONNECT userid    USERS = nnn
{DEV  rid  }
```

is the response to the primary system operator. DEV rid specifies the resource identification.

MESSAGE

Privilege Class: Any

Use the MESSAGE command to transmit message text to a specified userid or to the primary system operator. If the user designated to receive the message is not logged on or has suppressed the receiving of messages, the message is not transmitted and the sender receives a diagnostic message to this effect. A message which is not received by a user is not saved and must be sent at a later time when the user is receiving messages. The message is displayed at the terminal when the terminal is ready to receive output. If a typewriter terminal (or a display terminal having AUTOREAD set ON) is entering data, the class Any message is held until an end-of-line (carriage return or ENTER) signal is received. The format of the MESSAGE command is:

where:

userid is the identification of the single user who is to receive the message.

* specifies that you are sending a message to yourself.

OPERATOR sends the message to the primary system operator regardless of his userid.

msgtext is the text of the message which is to be transmitted. As many characters may be entered as will fit on the remainder of the input line.

Responses

hh:mm:ss
MSG FROM OPERATOR: msgtext

is the response received by the user from the system operator.

hh:mm:ss
MSG FROM { LOGONxxx }: msgtext
{ userid }

is the format of the message sent to another user or to the system operator, where userid is the name of the sender. If the user sending the message is not logged on to VM/370, LOGON and the line number are displayed instead of userid.

hh:mm:ss

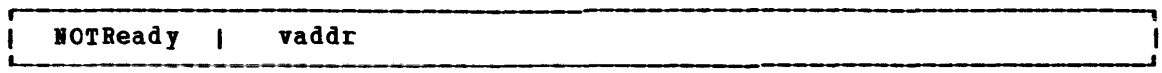
is the time in hours:minutes:seconds when the message was sent to the user.

If the user receiving the message is the primary system operator, the alarm bell at the central computer console rings.

NOTREADY

Privilege Class: G

Use the NOTREADY command to cause a virtual device to appear as if it had changed from ready to not ready status. This command is for use with spooled unit record devices and virtual consoles only. Any I/O operation to the specified device, in progress at the time the command is issued, is completed. On the next Start I/O (SIO) instruction, the not ready condition is in effect. The format of the NOTREADY command is:



where:

vaddr is the virtual device address (cuu) of the unit to be removed from ready status.

Response

INVALID DEVICE TYPE

This is the response if the device specified by vaddr is not a spooled unit record device or a virtual console.

ORDER

Privilege Class: G

Use the ORDER command to place your closed spool files, by device type, in a specific order. You can determine via the QUERY command the filename, filetype, originating userid, spoolid, and other attributes of all of your files. The files are ordered as they are passed to your spool device; you may order only your own files. The format of the ORDER command is:

```
 ORDer  | {Reader  } {CLass c1 CLass c2...}[1]
        | {Printer } {spoolid1 spoolid2...}
        | {PUnch   }
```

[1]Sequencing can be done with the ORDER command using a combination of CLASS and spoolid specifications. For example:

```
ORDER PRINTER CLASS A 1963 CLASS C
```

specifies that printer files are processed in the following order: all class A files, the file with spoolid 1963, and then all class C files.

where:

READER
RDR
: orders the reader spool files.

PRINTER
PRT
: orders the printer spool files.

PUNCH
PCH
: orders the punch spool files.

CLASS c1 CLASS c2...
: processes the input and output spool files in the order in which their classes are specified. CLASS is a required reserved word and c1, c2,... are one-character alphameric fields (with values from A to Z and from 0 to 9) representing the spooling classes.

spoolid1 spoolid2...
: processes the files represented by the spoolids in the order in which the spoolids are specified.

Response

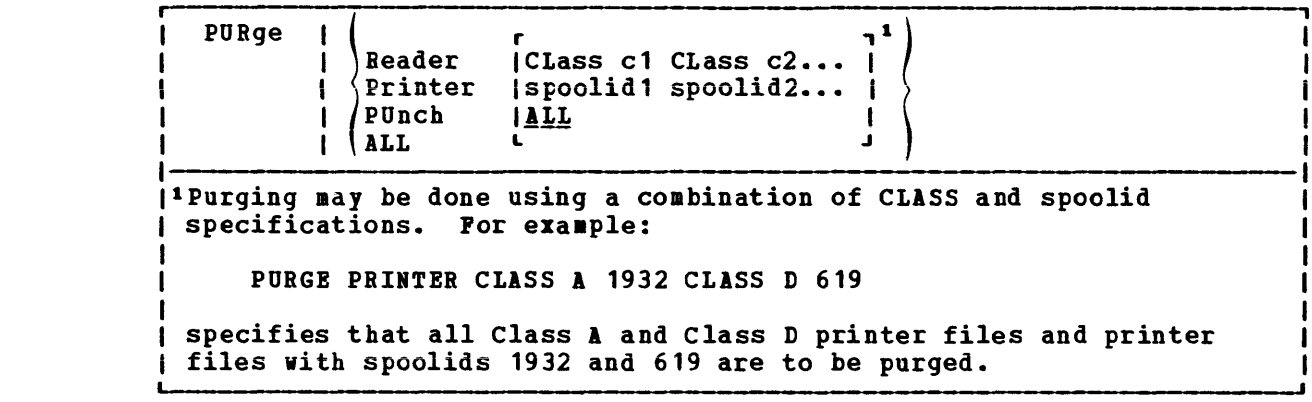
```
{nnnn} FILES ORDERED
{NO  }
```

This response indicates the number of files ordered. It is not displayed if you issued the CP SET IMSG OFF command.

PURGE

Privilege Class: G

Use the PURGE command to remove your own closed spool files from the system before they are printed or punched by the spooling devices, or before they are read by a user. Any closed file may be purged regardless of its status, as long as it has not been selected for processing. The format of the PURGE command is:



where:

READER
RDR
: purges all reader files.

PRINTER
PRT
: purges all printer spool files.

PUNCH
PCH
: purges all punch spool files.

ALL
: purges all spool files. When ALL is specified for device type, all other operands are ignored.

CLASS c1 CLASS c2...
: purges the files of the specified device type and class. CLASS is a required reserved word and c1, c2,... are one-character alphameric fields (with values from A to Z and 0 to 9) that represent the spooling class.

spoolid1 spoolid2...
: purges only the files for the specified spoolids.

ALL
: purges all files of the specified type (reader, printer, or punch).

Response

{nnnn / NO} FILES PURGED

This response indicates the number of files purged. It is not displayed if you issued the CP SET IMSG command.

QUERY

Privilege Class: G and all classes except class Any

Use the class G QUERY command to find the status of your system and machine configuration. You can request the following types of information:

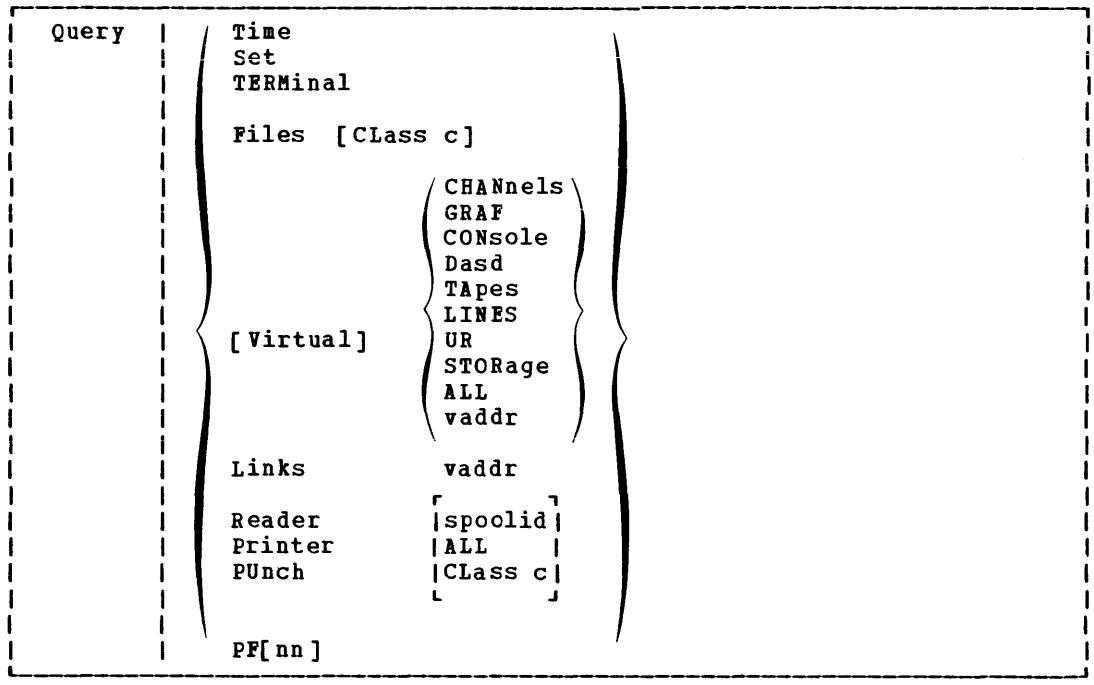
- How much time you have used during a terminal session.
- How many input and output spool files reside on your virtual machine.
- How you have set the functions of the SET command.
- How you have set the options of the TERMINAL command.
- The status of all the devices on your virtual machine.
- The channel operating mode of your virtual machine, either block-multiplexer or selector.
- All users, and their device addresses and access modes, who are linked to a given virtual address.
- Various kinds of information about your virtual printer, punch, and reader.

There are other operands you can use with the QUERY command if you have the privilege class required to use them. These are described in the VM/370: Operator's Guide. Also, if you are a CMS user, you can use the CMS QUERY command to query the status of your CMS virtual machine.

For ease of use, the QUERY command and operands described in this section have been separated into the operands available for general users (class G) and those available to all users except class Any.

The QUERY Command for Class G Users

The format of the Class G QUERY command is:



where:

TIME displays the current time, time zone, weekday, date, connect and CPU time for the current terminal session.

FILES [CLASS c]
displays the number of spooled input and output files for your virtual machine. Files currently being processed are not included in the totals. If CLASS is specified, the number of spooled input and output files of the class specified is displayed.

SET displays the status of the SET command functions.

TERMINAL displays the current options in effect for your virtual console environment.

VIRTUAL displays the status of all virtual devices.

CHANNELS displays the channel mode of operation for the virtual machine.

GRAF displays the status of all your virtual display devices that are locally attached.

CONSOLE displays the status of your virtual consoles.

DASD displays the status of all your virtual direct access storage devices.

TAPES displays the status of all your virtual magnetic tape devices.

LINES displays the status of all your virtual communication lines.

UR displays the status of all your unit record devices.

STORAGE displays the size of your virtual storage.

ALL displays the status of all your virtual devices.

vaddr displays the status of the virtual device at address vaddr.

LINKS vaddr
displays the userid, device address, and access mode at the terminal for all users linked to the specified virtual address (vaddr).

READER
RDR
PRINTER
PRT
PUNCH
PCH
displays the following information, pertaining to your virtual reader, virtual printer, and virtual punch spool files:

- Userid (of user who created the file)
- Spool file identification (spoolid)
- Class and originating device type
- Number of logical records in the file
- Number of copies specified for the file (has no effect for reader files)
- File hold status

One line of information is displayed for each spool file.

{READER / PRINTER / PUNCH} spoolid
displays additional information for one spool file. The spoolid operand must follow the READER, PRINTER, or PUNCH operand. In addition to the information normally displayed for reader, printer, or punch files, the following is also displayed:

- Date and time the file was created
- Filename and filetype of file (if any)
- Distribution code of the file (PRINTER and PUNCH files only)

Only one line of data is displayed (that data pertaining to the spool file specified by spoolid).

{FILES / READER / PRINTER / PUNCH} ALL
displays additional information for spool files. The ALL operand must follow the READER, PRINTER, or PUNCH operand. In addition to the information normally displayed for the reader, printer, or punch files, the following is also displayed:

- Date and time the file was created
- Filename and filetype of file (if any); if your file was assigned a dsname and you later issue QUERY, only the first 20 characters of the 24-character field are displayed.
- Distribution code of the file (PRINTER and PUNCH files only)

One line of information is displayed for each spool file of the type specified.

{FILES / READER / PRINTER / PUNCH} CLASS c

displays the basic information for all spool files of the class specified by c. This operand must follow the FILES, READER, PRINTER, or PUNCH operands.

One line of information is displayed for each spool file of the specified class.

PF[nn] displays the 3270 Program Function key number specified, along with its associated command lines. If nn is not specified, all 12 program function keys and their associated data lines are displayed. The value, nn, is a number from 1 (or 01) to 12. See the CP SET command for an explanation of how to define and use program function keys.

QUERY Command Responses for Class G Commands

This section describes the messages CP prints in response to your command.

QUERY TIME

TIME IS hh:mm:ss zone weekday mm/dd/yy

The current real clock time in hours:minutes:seconds, the time zone (for example, EST), the day of the week and the calendar date (month/day/year) are displayed.

CONNECT= hh:mm:ss VIRTCPU= mmm:ss.hs TOTCPU= mmm:ss.hs

The time spent in the current terminal session is displayed.

where:

CONNECT= hh:mm:ss is the actual clock time spent in the current terminal session in hours:minutes:seconds.

VIRTCPU= mmm:ss.hs is the virtual CPU time used in the current terminal session in minutes:seconds.hundredths of seconds.

TOTCPU= mmm:ss.hs is the total CPU time (virtual and overhead) used in the current terminal session in minutes:seconds.hundredths of seconds.

QUERY

QUERY SET

```
                                  ┌ON  ┐
MSG {ON } , WNG {ON } , EMSG      │OFF │
    {OFF}       {OFF}           ┌─┤CODE├─┐ , ACNT {ON } , RUN {ON }
                                └─┤TEXT├─┘        {OFF}       {OFF}
                                  └    ┘

LINEDIT {ON } , TIMER {ON  } , ISAM {ON } , ECMODE {ON }
        {OFF}         {OFF }       {OFF}         {OFF}
                      {REAL}

                 ┌ON  ┌SVC  ┐┐
ASSIST {         │    │     ├┤ , PAGEX {ON }
       {OFF      └    └NOSVC┘┘        {OFF}

IMSG {ON }
     {OFF}
```

The settings of all functions controlled by the SET command and the VM/370 directory ISAM and ECMODE options are displayed. Refer to the discussion of the SET command for explanations of the functions.

QUERY TERMINAL

```
LINEND {n  } , LINEDEL {n  } , CHARDEL {n  } , ESCAPE {n  }
       {OFF}           {OFF}           {OFF}          {OFF}

LINESIZE nnn, MASK {ON } , APL {ON } , ATTN {ON } , MODE {CP}
                   {OFF}       {OFF}       {OFF}      {VM}
```

The settings of all functions that are controlled by the TERMINAL command are displayed. Refer to the discussion of the TERMINAL command for explanations of the functions. If LINEDIT is turned off, the logical editing symbols displayed are those that were in effect before line editing was turned off.

QUERY FILES [CLASS c]

```
FILES: {nnn} RDR, {nnn} PRT, {nnn} PUN
       {NO }      {NO }      {NO }
```

The total number of spool files in your system is displayed. If you specify the CLASS option with QUERY FILES, only the totals for the class you specify are indicated rather than for all classes on your system.

QUERY VIRTUAL CHANNELS

```
CHANNELS= {SEL}
          {BMX}
```

The operating mode of the virtual machine channels is displayed. This reponse applies to all of the virtual machine channels except channel 0, which is always a byte-multiplexer channel, and any channels with virtual or real channel-to-channel adapters, which are always selector channels.

QUERY VIRTUAL GRAF

```
GRAF vaddr {ON DEV raddr}
           {NOT READY   }
```

The status of all locally attached virtual display devices defined to your virtual machine is displayed.

where:

vaddr is the virtual address to which the device is attached.

raddr is the real address of the device.

NOT READY shows the status of a virtual display device that has not been attached via the DIAL command.

QUERY VIRTUAL CONSOLE

```
                 {GRAF}          {  TERM  }{STOP  }
CONS vaddr ON    {LINE} raddr    {NOTERM  }{START }

                 {  CONT  }{  HOLD  }          {   READY  }
vaddr CL c       {NOCONT  }{NOHOLD  } COPY nn  {NOTREADY  }

           {TO }
vaddr      {FOR} userid    DIST distcode
```

For virtual machine consoles, a three-line response is displayed. The first line shows the console status and options and the next two lines are the virtual console spooling status.

where:

vaddr is the virtual address of the virtual machine console.

raddr is the real address of the terminal associated with the virtual console.

c is the spooling class of the console.

nn is the number of copies spooled.

userid is the user identification.

distcode is the distribution code.

The other fields indicate the setting of the respective options in the SPOOL command.

The default settings for a virtual console are:

```
CONS vaddr ON DEV raddr  TERM STOP
     vaddr CL T NOCONT NOHOLD COPY 01 READY
     vaddr FOR userid  DIST distcode
```

QUERY VIRTUAL DASD

DASD vaddr type volser {R/W / R/O} nnn CYL

The status of each virtual disk defined for your system is displayed.

where:

vaddr is the virtual address to which the DASD device is attached.

type is one of the following device types:

2311
2305
2314
3330
3340
231T (2311 at top of 2314)
231B (2311 at bottom of 2314)

volser is the volume serial number of the system disk on which this virtual disk resides.

R/W
R/O indicates the read/write status of the disk.

nnn is the number of cylinders on the virtual disk.

QUERY VIRTUAL TAPES

TAPE vaddr ON DEV raddr

The status of each tape defined for your system is displayed.

where:

vaddr is the virtual address to which the tape is attached.

raddr is the real address of the tape.

QUERY VIRTUAL LINES

LINE vaddr ON DEV raddr

The status of all communication lines defined in your virtual machine is displayed.

where:

vaddr is the virtual address to which the line is attached.

raddr is the real address of the line.

LINE vaddr {ENABLED / DISABLED}

The status of virtual communication lines at virtual address vaddr is displayed.

QUERY VIRTUAL UR

```
                                 { CONT   } { HOLD   } { EOF   } {  READY   }
RDR vaddr CL c                   { NOCONT } { NOHOLD } { NOEOF } { NOTREADY }
```

The status of all the virtual readers attached to your virtual machine is displayed.

where:

vaddr is the virtual device address of the virtual reader.

c is the spool file class which the device services. A class of * indicates the device serves all classes of spool files for input.

The other fields indicate the setting of the respective options in the SPOOL command.

The default settings for a reader are:

```
RDR vaddr CL * NOCONT NOHOLD READY EOF
```

```
{ PRT } vaddr CL c { CONT   } { HOLD   } COPY nn { READY   }
{ PUN }            { NOCONT } { NOHOLD }        { NOTREADY }

  vaddr { TO  } userid   DIST distcode
        { FOR }
```

The status of all the virtual printers and punches attached to your virtual machine is displayed.

where:

vaddr is the virtual device address of the virtual printer or punch.

c is the output class assigned to spool files produced from the device.

nn is the number of copies of each output file to be produced.

TO userid indicates that the output from the device, when closed, becomes a reader input spool file for the indicated userid.

FOR userid indicates the userid identification (spool file owner) assigned to spool files produced from the device.

distcode is the distribution code assigned to each spool file produced from the device.

Note: The distcode in this case indicates the FOR userid; however, the distcode produced on the output files when the file is closed is the distcode assigned to the FOR userid as specified in the VM/370 directory.

The other fields indicate the setting of the respective options in the SPOOL command.

The default settings are:

```
{PRT}
{PUN} vaddr CL A NOCONT NOHOLD COPY READY 01
      vaddr  FOR userid  DIST distcode
```

where:
userid and distcode are assigned for the virtual machine.

QUERY VIRTUAL STORAGE

STORAGE = nnnnnK

The size of the virtual machine in multiples of 1024 bytes is displayed.

QUERY VIRTUAL ALL

Has the same effect as if all the following commands were issued:

```
QUERY VIRTUAL STORAGE
QUERY VIRTUAL LINES
QUERY VIRTUAL TAPE
QUERY VIRTUAL UR
QUERY VIRTUAL DASD
QUERY VIRTUAL GRAF
QUERY VIRTUAL CONSOLE
QUERY VIRTUAL CHANNELS
```

QUERY VIRTUAL vaddr

The response is in the same form as QUERY VIRTUAL DASD, TAPES, LINES, or UR, depending on virtual device type.

QUERY LINKS vaddr

```
userid vaddr {R/O}, ...
  .      .   {R/W}
  .      .
  .      .
```

A list of users who linked to the device at virtual address vaddr is displayed.

where:

userid — is the identification of the user who originated the link.

vaddr — is the virtual address by which the user (userid) refers to the device.

R/O
R/W — is the type of access the user (userid) has to the device.

QUERY READER, QUERY PRINTER, QUERY PUNCH

```
ORIGINID FILE  CLASS RECDS  CPY HOLD |DATE  TIME     NAME TYPE DIST    |
userid spoolid c typ norecs nn  stat |mm/dd hh:mm:ss fn   ft   distcode|
                                     L                                 J
    .      .  .  .   .     .   .      .       .      .    .      .
    .      .  .  .   .     .   .      .       .      .    .      .
    .      .  .  .   .     .   .      .       .      .    .      .
```

where:

userid is the user who originally created the file.

spoolid is a unique, system-assigned number which is used by VM/370 to identify the file.

c is the spool file class.

typ is the originating device type (PRT, PUN, CON, or RDR).

norecs is the number of logical records contained in the file.

nn is the number of copies assigned to the file (it has no effect for virtual reader files).

stat is the file hold status: NONE (no hold), USER (user hold), SYS (system hold), or USYS (system and user hold).

mm/dd is the date the file was created in month/day.

hh:mm:ss is the time of file creation in hours:minutes:seconds.

fn is the filename assigned to the file (if any).

ft is the filetype assigned to the file (if any).

distcode is the distribution code assigned to the file.

When you issue QUERY READER, QUERY PRINTER, or QUERY PUNCH commands, CP responds by listing (in the form described) all the files associated with your virtual reader, printer, or punch.

The information listing DATE, TIME, NAME, TYPE, and DIST (date of file creation, time of file creation, name of file, filetype of file, and file distribution code) is displayed only when you specify the ALL or spoolid operands.

QUERY PFnn

```
      {IMMED }
PFnn  {DELAY } pfdata1...
```

The program function defined for a program function key is displayed. If there is no function defined for the program function key, this message is generated in the user input area of the screen:

PFnn UNDEFINED

Note: If the next command you enter is shorter than this message, you must first clear the input area or enter enough blanks to eliminate the message; otherwise, errors result.

QUERY Command for All Classes of Users (Except Class Any)

Use this form of the QUERY command to:

- Display the log messages.
- List all the users that are logged on.
- Display the number of users that are logged on or dialed to VM/370.

This form of the QUERY command is for all classes of users except those in the Any category. The format for this QUERY command is:

```
Query | LOGmsg
      | Names
      | Users   [userid]
      | userid
```

where:

LOGMSG — displays the log messages of the day.

NAMES — displays a list of all the users logged on and the real address of the line to which each is connected. If a user is disconnected, DSC is printed instead of the line address.

USERS — displays the number of logged on users and the number of users logically connected to other virtual machines.

USERS userid
userid — displays the user identification and the terminal device address of the specified user if he is logged on. If the user is not logged on, a message to this effect is issued. Use the QUERY USERS userid format if the userid is the same as an operand of the QUERY command (for example, TAPES).

QUERY Command Responses for All Classes of Commands

QUERY LOGMSG

```
* logmsg text line 1
  .             .
  .             .
  .             .
* logmsg text line n
```

All lines (both those with an asterisk and without) in the log message file are displayed.

QUERY NAMES

```
userid - {DSC   }, ...
         {raddr }
  .
  .
  .
userid - {DSC   }, ...
         {raddr }
```

A list of all logged-on users is displayed; if the user is currently connected, the real address to which he is connected is displayed (raddr); if he is not connected to the system, DSC is displayed.

QUERY USERS

nnn USERS, mmm DIALED

The number of users logged on and dialed to VM/370 is displayed.

where:

nnn is the total number of logged-on users.

mmm is the total number of users attached via DIAL to virtual machines.

Note: DIALED means the line is not available to CP because it is logically attached to a multiple-access virtual machine and is a part of that user's virtual machine operation.

QUERY userid or QUERY USERS userid

userid - raddr

The real address (raddr) to which the specified user is connected is displayed.

READY

Privilege Class: G

Use the READY command to set a device-end interrupt pending for the specified virtual device. The status of the virtual machine is unchanged. Other than having a device-end interrupt pending, the virtual device is unchanged. The format of the READY command is:

where:

vaddr is a virtual device address (cuu).

Responses

None.

REQUEST

Privilege Class: G

Use the REQUEST command to make an attention interrupt pending at your virtual console. The format of the REQUEST command is:

```
| REQuest |
```

The ATTN command performs the same functions as REQUEST and the two commands can be used interchangeably.

Responses

None.

RESET

Privilege Class: G

Use the RESET command to clear all pending interrupts from the specified virtual device. In addition, all error conditions occurring as a result of unit checks and virtual sense bytes are reset. The format of the RESET command is:

where:

vaddr is a virtual device address (cuu) of the device to be reset.

Responses

DEVICE RESET

REWIND

Privilege Class: G

Use the REWIND command to rewind (but not unload) a real tape unit attached to your virtual machine at a specific virtual device address. This accomplishes the manual operation of rewinding and making the tape ready at the tape unit. The format of the REWIND command is:

```
┌──────────────────────────────────────────────────────────────────┐
| REWind | vaddr                                                   |
└──────────────────────────────────────────────────────────────────┘
```

where:

vaddr is the virtual device address (cuu) of the tape unit to be rewound.

Responses

REWIND COMPLETE

This is the normal response.

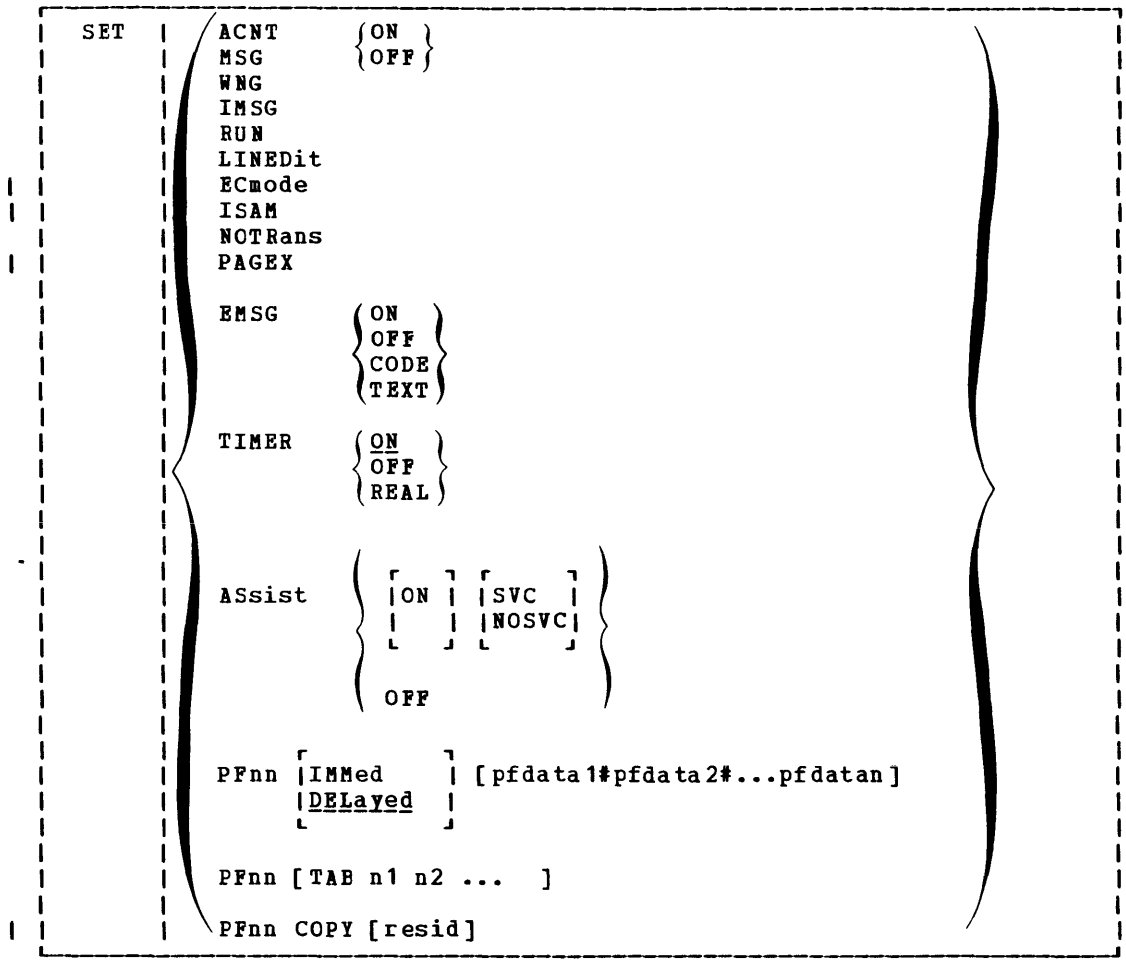
REWIND NOT PERFORMED

This is the response you receive if the real tape unit is not ready.

SET

Privilege Class: G

Use the SET command to control various functions within your virtual system. The format of the SET command is:



where:

ACNT {ON / OFF} controls whether accounting information is displayed at the terminal or not (ON and OFF respectively) when the operator issues the CP ACNT command. When you log on VM/370, ACNT is set on.

MSG {ON / OFF} controls whether messages sent by the MSG command from other users are to be received at the terminal. If ON is specified, the messages are displayed. OFF specifies that no messages are received. When you log on VM/370, MSG is set on.

WNG {ON / OFF} controls whether warning messages are displayed at the terminal. If ON is specified, all warning messages sent

via the CP WARNING command from the system operator are received at the terminal. If OFF is specified, no warning messages are received. When you log on VM/370, WNG is set on.

IMSG {ON | OFF} controls whether certain informational responses issued by the CP CHANGE, DEFINE, DETACH, ORDER, PURGE, and TRANSFER commands are displayed at the terminal or not. The descriptions of these CP commands tell which responses are affected. If ON is specified the informational responses are displayed. If OFF is specified, they are not. The SET IMSG ON or OFF command line has no effect on the handling of error messages set by the SET EMSG command. When you log cn VM/370, IMSG is set on.

RUN {ON | OFF} controls whether the virtual machine stops when the Attention key is pressed. ON allows you to activate the Attention key (causing a read of a CP command) without stopping your virtual machine. When the CP command is entered, it is immediately executed and the virtual machine resumes execution. OFF places the virtual machine in the normal CP environment, so that when the Attention key is pressed, the virtual machine stops. When you log on VM/370, RUN is set off.

LINEDIT {ON | OFF} controls the line editing functions. ON specifies that the line editing functions and the symbols of the VM/370 system are to be used to edit virtual CPU console input requests. This establishes line editing features in systems that do not normally provide them. OFF specifies that no character or line editing is to be used for the virtual machine operating system. When you log on VM/370, LINEDIT is set on.

ECMODE {ON | OFF} controls whether the virtual machine operating system may use System/370 extended control mode and control registers 1 through 15. Control register zero may be used with ECMODE either ON or OFF. When you log on VM/370, ECMODE is set according to the user's directory option; ON if ECMODE was specified and OFF if not.

Note: Execution of the SET ECMODE {ON|OFF} command always causes a virtual system reset.

ISAM {ON | OFF} controls whether additional checking is performed on virtual I/O requests to DASD in order to support the use of the OS Indexed Sequential Access Method (ISAM). When you log on VM/370, ISAM is set according to the user's directory options; ON if ISAM was specified and OFF if not.

NOTRANS {ON | OFF} controls CCW translation for CP. NOTRANS can be specified only by a virtual machine that occupies the virtual=real space. It causes all virtual I/O from the issuing virtual machine to bypass the CP CCW translation. To be in effect in the virtual=real

environment, SET NOTRANS ON must be issued after the virtual=real machine is loaded via the IPL command. (IPL sets the NOTRANS option to an OFF condition.)

PAGEX {ON / OFF} controls the pseudo page fault portion of the VM/VS Handshaking feature. PAGEX ON or OFF should only be issued for an OS/VS1 virtual machine that has the VM/VS Handshaking feature active. It can only be specified for a virtual machine that has the extended control mode (ECMODE) option. PAGEX ON sets on the pseudo page fault portion of handshaking; PAGEX OFF sets it off. When you log on to VM/370, PAGEX is set OFF.

EMSG {ON / OFF / CODE / TEXT} controls error message handling. ON specifies that both the error code and text are displayed at the terminal. TEXT specifies that only text is displayed. CODE specifies that only the error code be displayed. OFF specifies that no error message is to be displayed. When you log on VM/370, EMSG is set to TEXT.

Note, CMS recognizes EMSG settings for all error (E), information (I), and warning (W) messages, but ignores the EMSG setting and displays the complete message (error code and text) for all response (R), severe error (S), and terminal (T) messages.

TIMER {ON / OFF / REAL} controls the virtual timer. ON specifies that the virtual timer is to be updated only when the virtual CPU is running. OFF specifies that the virtual timer is not be updated. REAL specifies that the virtual timer is to be updated during virtual CPU run time and also during virtual wait time. If the REALTIMER option is specified in your VM/370 directory entry, TIMER is set to REAL when you lcg on; otherwise it is set to ON when you log on.

ASSIST { [ON] [SVC / NOSVC] / OFF }

controls the availability of the virtual machine assist feature for your virtual machine. The assist feature is available to your virtual machine when you log on if (1) the real CPU has the feature installed and (2) the system operator has not turned the feature off. The SVC handling portion of the assist feature is invoked when you log on unless your VM/370 directory entry has the SVCOFF option. Issue the QUERY SET command line to see if the assist feature is activated and whether the assist feature or VM/370 is handling SVC interrupts.

All SVC 76 requests are passed to CP for handling, regardless of the SVC and NOSVC operands.

If you issue the SET ASSIST command line and specify SVC or NOSVC while the virtual machine assist feature is turned off, the appropriate bits are set. Later, if the feature is turned on again, the operand you specified while it was off becomes effective.

ON sets the assist feature on for the virtual machine; OFF turns it off. SVC specifies that the assist feature handles all SVC interrupts except SVC 76 for the virtual machine; NOSVC means VM/370 handles the SVC interrupts. See the VM/370: System Programmer's Guide for information on how to use the assist feature.

```
PFnn ┌IMMED   ┐ [pfdata1#pfdata2#...pfdatan]
     |DELAYED |
     └        ┘
```

defines a program function for a program function key on a 3277 Display Station and indicates when that function is to be executed. See the VM/370: Terminal User's Guide for a description of how to use the 3277 program function keys.

The value, nn, is a number from 1 (or 01) to 12 that corresponds to a key on a 3277. The program function is a "function", or programming capability, you create by defining a series of VM/370 commands or data you want executed. This series of commands executes when you press the appropriate program function key.

IMMED specifes that the program function is executed immediately after you press the program function key.

DELAYED specifies that execution of the program function is delayed for a display terminal. When the program function is entered, it is displayed in the input area and not executed until you press the Enter key. DELAYED is the default value for display terminals.

pfdata1#pfdata2#...pfdatan defines the VM/370 command or data lines that constitute the program function. If more than one command line is to be entered, the pound sign (#) must separate the lines. If you use the pound sign (#) to separate commands that you want executed with the designated PF key, you must precede the command line with #CP, turn line editing off, or precede each pound sign with the logical escape character ("). For further explanation, see the "Examples of Setting Program Function Keys" section that follows. If no command lines are entered, PFnn is a null command. Program functions cannot be embedded within one another.

PFnn TAB n1 n2 ...
specifies a program function number to be associated with tab settings on a terminal. The number of the PF key, nn, can be a value from 1 (or 01) to 12. See the VM/370: EDIT Guide for examples of how this feature is used.

TAB is a keyword identifying the tab setting function. The tab settings may be entered in any order.

| PFnn COPY [resid]
| specifies that the program function key, numbered nn, performs a COPY function for a remote 3270 terminal. nn must be a value of 1 or 01 to 12. The COPY function produces a printed output of the entire screen display at the time the PF key is actuated. The output is printed on an IBM 3284, 3286 or 3288 printer connected to the same control unit as your display terminal.

The resid operand may be specified if more than one printer is connected to the same control unit as your display terminal. It is a three-character hexadecimal resource identification number assigned to a specific printer. If resid is entered, the printed copy is directed to a specific printer; if not, the copy is printed on the printer with the lowest resid number. The resid numbers of the printers available to your display terminal can be obtained from your system operator. If only one printer is available, resid need not be specified.

If the command is invalid or if the designated or default printer is not free (other display terminals may be using it) or is not connected to the same control unit as your display terminal, a NOT ACCEPTED message appears on the screen, If the printer was busy, retry the operation until the printer honors your request.

You may include your own identification on the printed output by entering the data into the user input area of the screen before you press the PF key. The identification appears in the lower left of the printed copy.

Examples of Setting Program Function Keys

This example shows you how the SET PFnn command is processed if you do not turn line editing off or use the logical escape character.

Enter one of the following commands while in CMS mode:

```
SET PF02 IMMED Q RDR#Q PTR#Q PUN

          -- or --

CP SET PF02 IMMED Q RDR#Q PTR#Q PUN
```

Now press the ENTER key:

1. The ENTER key causes immediate execution,
2. Only the Q PTR and Q PUN commands execute, and
3. Q PTR and Q PUN are stripped from the PF02 key assignment leaving Q RDR, which was not executed.

The following examples demonstrate two methods for avoiding the problem.

Example 1

Enter one of the following commands while in CMS mode:

```
#CP SET PF02 IMMED Q RDR#Q PTR#Q PUN

            -- or --

CP SET PF02 IMMED Q RDR"#Q PTR"#Q PUN

            -- or --

SET PF02 IMMED Q RDR"#Q PTR"#Q PUN
```

Now press the ENTER key.

CP assigns the three QUERY commands as functions of the PF02 key. Pressing the PF02 key executes the three QUERY commands.

Example 2

Enter the following command while in CMS mode:

```
SET LINEDIT OFF
```

and press the ENTER key.

Then enter:

```
SET PF02 IMMED Q RDR#Q PTR#Q PUN

            -- or --

CP SET PF02 IMMED Q RDR#Q PTR#Q PUN
```

and press the ENTER key.

CP assigns the three QUERY commands as functions of the PF02 key.

Then enter:

```
SET LINEDIT ON
```

and press the ENTER key.

Pressing the PF02 key executes the three QUERY commands.

Response

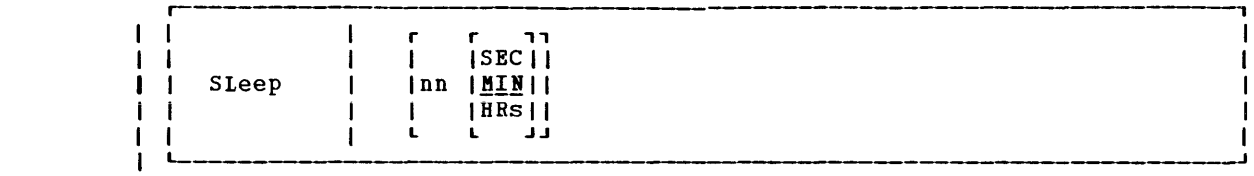
* PFnn UNDEFINED

This response appears in the user area of the screen on a 3277 Display Station if a PF key that is undefined is pressed.

SLEEP

Privilege Class: Any

Use the SLEEP command to place the virtual machine in a dormant state but allow messages to be displayed. The virtual machine does not run during this time, but connection time is still being counted. You can specify a sleep interval in the command line and the virtual machine is awakened automatically when the specified interval has elapsed. Awaken the terminal at anytime by signalling attention. In either case, this returns the virtual machine to the environment from which SLEEP was issued. If no interval is specified, the virtual machine remains dormant until awakened by signalling attention. The format of the SLEEP command is:



where:

indicates the actual number of seconds, minutes, or hours of actual CPU time to sleep. The value nn can be any decimal number from 00 through 99. If you specify no time unit, the value of nn is taken to be minutes.

Usage:

If you issue the SLEEP command from a CP read or from a VM read using the CP "escape" function (#CP SLEEP), the end of the time interval or signalling attention returns you to the CP environment.

If you issue the SLEEP command while in virtual machine mode (for example, CMS execution of the command line CP SLEEP), the end of the time interval or signalling attention returns your terminal to virtual machine mode without entering the CP environment.

The SLEEP command, with the time interval, is a convenient way to delay or schedule the execution of certain jobs that could be run more efficiently at a later time; for example, second shift.

Responses

None.

SPOOL

Privilege Class: G

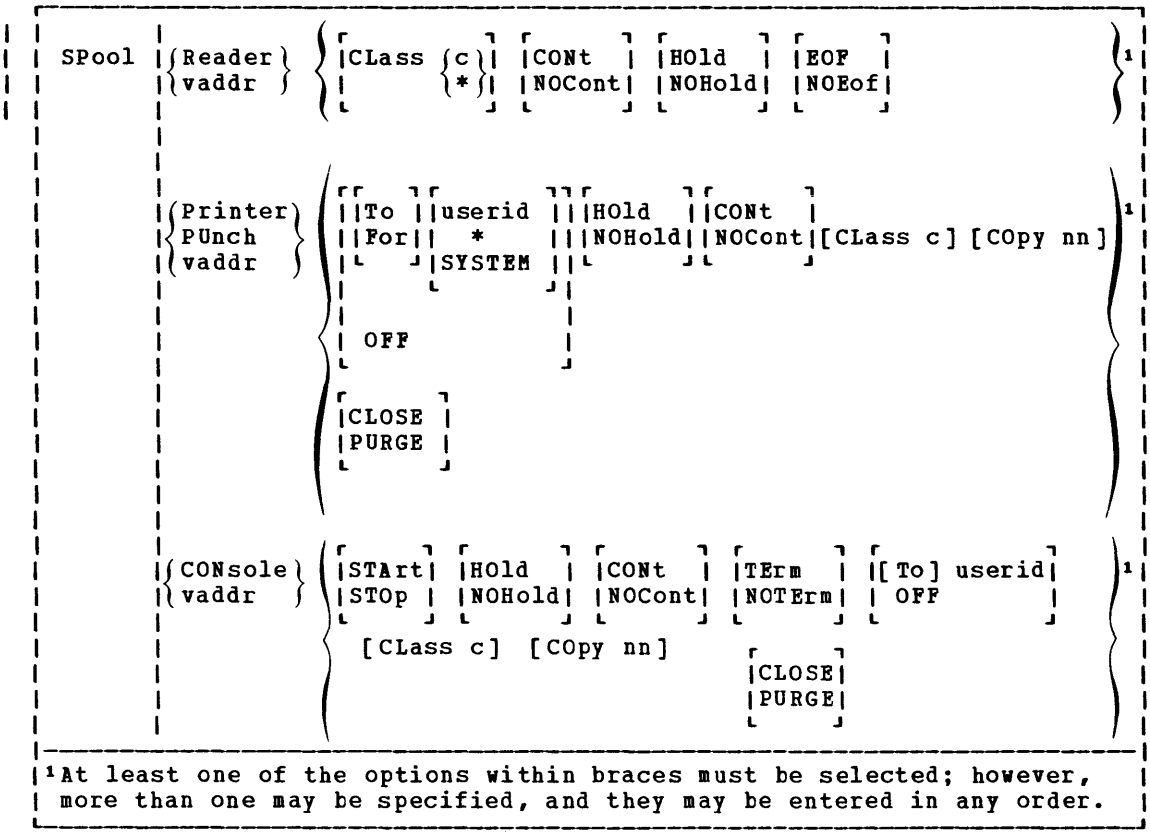
Use the SPOOL command to modify the spooling control options in effect for a given virtual spooling device or for a group of devices. The SPOOL command can also initialize or stop the spooling of virtual console input and output.

You can direct a file to a remote location by using the SPOOL command in conjunction with the TAG command. The section "Transmitting Files to Remote Locations" which follows discusses the form of the SPOOL command you use to spool files across the Remote Spooling Communications Subsystem (RSCS).

Unless otherwise set, the following options are default values for spool files:

Spool File	Options
Reader	NOHOLD, NOCONT, EOF, CLASS * as specified in the VM/370 directory entry
Printer Punch	OFF, NOHOLD, NOCONT, COPY 01, CLASS as specified in the VM/370 directory entry
Console	NOHOLD, NOCONT, TERM, OFF, CLASS T, COPY 01

The format of the SPOOL command is:



where:

READER
RDR
: modifies the options for all reader spool files.

PRINTER
PRT
: modifies the options for all printer spool files.

PUNCH
PCH
: modifies the options for all punch spool devices.

CONSOLE
: modifies the options for the virtual console spool file and/or initiates or stops the spooling of virtual console input and output, including CP input/output.

vaddr
: is the device address (cuu) of the virtual unit record device or console whose options are to be modified.

CLASS { c / * }
: specifies the spool class of the device. The c is a one-character alphameric field whose values can be A through Z, 1 through 9, or * (asterisk).

  Unless your virtual reader class is asterisk (*), you must ensure that any files to be read by your virtual reader are of the same spool class as your virtual reader. The * is the universal class; if your virtual reader is class *, it can read any file, regardless of class.

CONT
: ignores intermediate end-of-file indicators or CLOSE requests. For virtual readers, reading is continuous with all end-of-file indicators ignored until all files spooled to the virtual machine are read in. If this option is not in effect, a unit exception is reflected to the virtual machine at the end of each spooled file. CONT specified for the punch or printer causes all CLOSE requests to be ignored until reset by NOCONT. If CONT is specified, NOCONT cannot be specified.

  CONT specifies that reading is to continue without intervening end-of-file indications until all files in the system that belong to the user are read. If CONT is not in effect or is reset by specification of NOCONT, an end-of-file indication is reflected to the virtual machine at the end of each SPOOL file in the system. The nature of the end-of-file indication to be reflected is set by the EOF and NOEOF options. If the EOF option is in effect, end-of-file is signaled by a unit exception: this corresponds to pressing the end-of-file button on a real card reader. If NOEOF is in effect for a virtual reader, end-of-file is signaled by the reflection of a unit check/intervention required status.

NOCONT
: resets the continuous spooling option. If NOCONT is specified, CONT cannot be specified.

HOLD
: places all files created by the specified device in a user HOLD status. For READER files, this option specifies that input files for the specified reader are not deleted from the system after they are read. The status of all files must be changed by the CHANGE command. The status of output devices is changed by the SPOOL command. If HOLD is specified, NOHOLD cannot be specified.

If the HOLD option is specified for a virtual printer or punch that is transferred to a user for input (TO userid), that virtual device places a user HOLD status on the reader file. The user receiving the file cannot read its status until it is changed by issuing the CHANGE command with the NOHOLD operand. The spool file class of the virtual output device must match the class of the receiver's virtual reader (or the virtual reader must have a class of *) in order for the spool file to be processed. If these conditions are not satisfied, the reader appears empty to the virtual machine attempting to read a file, even though reader files do exist.

If a virtual reader is operating with CONT and HOLD, then virtual reader files are saved and placed in a user HOLD status. The file cannot be read until it is changed (using the CHANGE command) to a NOHOLD status.

NOHOLD resets the HOLD operand. Future files are not held. NOHOLD resets the HOLD operand in effect for the specified reader. This operand can be overridden for an active file being closed by the CLOSE command using the HOLD or NOHOLD operand. If NOHOLD is specified, HOLD may not be specified.

EOF sets a virtual end-of-file condition on the specified reader, thereby ensuring that a unit exception condition is reflected on the read that follows the reading of the last card in a file. If EOF is specified, NOEOF may not be specified.

NOEOF specifies that the reading continues to physical end-of-file. The virtual reader stops when no cards are left in the reader and when a unit check/intervention required status is pending. If NOEOF is specified EOF may not be specified.

[TO] userid
*
SYSTEM

transfers the output of the virtual device to the virtual card reader of the specified userid. If TO is omitted, the userid may not be TO or T. TO * may be coded if the output is to be transferred to your own virtual card reader. If TO userid is specified, neither OFF nor FOR may be specified on the same command line.

If you specify COPY with TO userid, the number of copies you specify has no effect on the receiver of the spool file; he receives only one copy. However, if OFF or FOR are specified on a subsequent command, the receiver of your spool file receives the number of copies you specify via COPY. For example, if the following command is entered:

```
SPOOL PUN TO USERA COPY 3 CLASS B
```

the COPY operand has no effect on the file going to USERA. However, if the command:

```
SPOOL PUN OFF
```

is entered following the first command, the COPY 3 specified in the first statement effects the second command.

TO SYSTEM is equivalent to specifying OFF and resets the transferred spool option.

[FOR] userid
: indicates the userid under which printed or punched output is produced. The userid becomes the owner of the output spool file and the distcode on the file is the distcode for the user that is specified in the VM/370 directory. The file is not transferred to the user's reader input. The default setting is for your own virtual machine identification. FOR *, or FOR SYSTEM can be coded to specify your own identification and is equivalent to the OFF option.

OFF
: resets the transferred spool option.

COPY nn
: is the number of copies that are to be printed or punched when the file is spooled to the real unit record equipment. This operand is valid only for output files; the number of copies, nn must be between 1 and 99 (leading zeros need not be specified).

CLOSE
: closes the specified device regardless of the CONT setting for the device. If CLOSE is specified, PURGE may not be specified. CLOSE does not affect the setting of any other operand and is provided as a convenience to close a virtual output device. As an example, this sequence of commands:

```
SPOOL PRT CONT
  (print file)
  (print file)
  (print file)
SPOOL PRT NOCONT
CLOSE PRT
SPOOL PRT CONT
  (print file)
  (print file)
        .
        .
        .
```

: can be replaced with the following sequence to achieve the desired result:

```
SPOOL PRT CONT
  (print file)
  (print file)
  (print file)
SPOOL PRT CLOSE
  (print file)
  (print file)
        .
        .
        .
```

| PURGE
: closes and purges the spool file from the specified virtual output device regardless of the CONT setting for the device. If PURGE is specified, CLOSE cannot be specified. PURGE does not affect the setting of any other operand and is equivalent to issuing the CLOSE command for a device (or type of device) with the PURGE operand. This form of the SPOOL command is provided for your convenience.

START
: places all console input and output in a spool file. Until a CLOSE is issued for the console, characteristics of the console spool file may be changed by use of the SPOOL CONSOLE command. After the console is closed, the file becomes a

printer spool file whose characteristics can be changed by issuing the CHANGE PRINTER command.

STOP terminates the spooling of console input and output. The command SPOOL CONSOLE STOP does not close the console spool file.

TERM displays the virtual console input and output at the terminal in addition to placing it in a spool file. The TERM operand has no effect until the START operand is specified.

| NOTERM suppresses the display of console input and output of a system running in a virtual machine. The display of console input and output of CP console functions, entered from CP mode, are not suppressed. The NOTERM operand has no effect until the START operand is specified.

The Spoolid Number: A Unique Identifier for Your Spool Files

Once you close a spool file by issuing the CMS PRINT or PUNCH command or the CP CLOSE command, CP assigns the spool file a number between 1 and 9900. This number is called the spoolid (spool file identification) for the file. It can be used as a convenient way to uniquely identify the file. It can also be used when you are manipulating the file with VM/370 spooling commands such as ORDER, CHANGE or CLOSE.

Spoolids are assigned to all your spool files sequentially. When the maximum number (9900) is assigned, CP begins again with the number 1.

When you print or punch a file, CP displays at your terminal the spoolid it assigned to your file. You can find out various kinds of information about a file using the spoolid with the many forms of the QUERY command.

Transmitting Files to Remote Locations

To direct files to remote stations, use the CP TAG and SPOOL commands in conjunction with a command that causes the file to be closed and sent to a virtual device (for example, a virtual printer or punch). Use the TAG command to specify the device to be spooled and to associate with that device the location identifier for the destination of the file:

```
TAG DEV device locid
```

where device is the virtual device type (for example, PRINTER or PUNCH) or virtual device address (vaddr) and locid is the name of the destination to which the file is to be transmitted.

Use the SPOOL command to specify that output to the device specified in the command is to be sent to the RSCS virtual machine, which perfcrms the actual transmission of the file:

```
SPOOL device TO userid
```

where device is the same virtual device type or virtual device address specified in the TAG command and userid is the userid of the RSCS virtual machine at your installation. You can find out the userid of your installation's RSCS virtual machine and the locid for the various remote stations from your installations's system programmer.

After you issue the TAG and SPOOL commands, use a command (such as the CMS PRINT or PUNCH command or the CP CLOSE command) to cause the spool file to be generated, closed, and spooled to the specified virtual device.

The following example shows how to use these three commands to transmit a CMS file to a remote location:

```
TAG DEV PUNCH CAMBRIDG

SPOOL PUNCH TO NET

PUNCH MYPROG ASSEMBLE
```

The TAG command defines the type of file to be transmitted, a punch file, and the remote station to which you want it transmitted, CAMBRIDG. NET is the userid of the virtual machine controlling the RSCS network: you direct your file to that virtual machine with the SPOOL command. The PUNCH command causes the file MYPROG ASSEMBLE to be punched on your virtual machine card punch, closed, and then spooled to the virtual reader of the RSCS virtual machine, which you specified in the SPOOL command. The RSCS virtual machine then processes your file (now a VM/370 spool file) and transmits it across the RSCS network.

Receiving Files from the RSCS Network

If your virtual machine is logged on VM/370, RSCS notifies you of the arrival of a file for your machine from the RSCS network by displaying a message at your terminal. The file is sent to your virtual card reader.

VM/370 can accumulate files from the RSCS network destined for your virtual card reader, regardless of whether you are logged on your virtual machine or not. If you are logged on your virtual machine, issue the QUERY command to see if you have any files in your virtual reader. When you log on your virtual machine, the logon process transmits a message informing you of accumulated spool files residing in your virtual reader (punch or printer).

Responses

None.

STORE

Privilege Class: G

Use the STORE command to alter the contents of specified registers and locations of the virtual machine. The contents of the following can be altered:

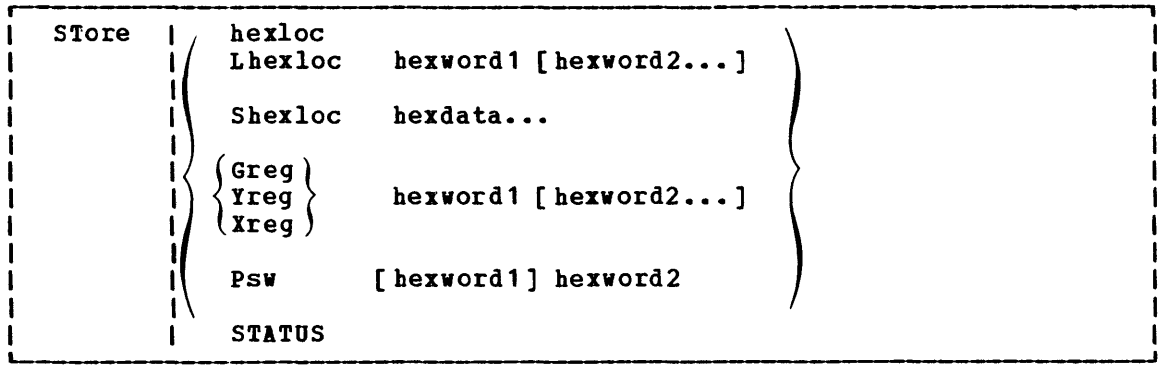
- Virtual storage locations
- General registers
- Floating-point registers
- Control registers (if available)
- Program status word

The STORE command can also save virtual machine data in low storage.

The operands may be combined in any order desired, separated by one or more blanks, for up to one full line of input. If an invalid operand is encountered, an error message is issued and the store function is terminated. However, all valid operands entered, before the invalid one, are processed properly.

Storage locations, registers, the PSW, and status can be stored using a single command line. When you combine the operands for storing into storage, registers, the PSW, or the status area on a single command line, all operands must be specified; default values do not apply in this case.

The format of the STORE command is:



where:

hexloc
Lhexloc hexword1 [hexword2...]
stores the specified data (hexword1 [hexword2...]) in successive fullword locations starting at the address specified by hexloc. The smallest group of hexadecimal values that can be stored using this form is one fullword. Alignment is made to the nearest fullword boundary. Either form (hexloc or Lhexloc) can be used.

The operands (hexword1 hexword2...) each represent up to eight hexadecimal digits. If the value being stored is less than a fullword (eight hexadecimal digits), it is right-adjusted in

the word and the high order bytes of the word are filled with zeros. If two or more hexwords are specified, they must be separated by one or more blanks.

Shexloc hexdata...
stores the data specified (hexdata...) in the address specified by hexloc, without word alignment. The shortest string that can be stored is one byte (two hexadecimal digits). If the string contains an odd number of characters, the last character is not stored, an error message is sent, and the function is terminated.

The operand, hexdata, is a string of two or more hexadecimal digits with no embedded blanks.

Greg hexword1 [hexword2...]
stores the hexadecimal data (hexword1 [hexword2...]) in successive general registers starting at the register specified by reg. The reg operand must be either a decimal number from 0-15 or a hexadecimal digit from 0-F.

The operands (hexword1 [hexword2...]) each represent up to eight hexadecimal digits. If less than eight digits are specified, the string is right justified in a fullword and left-filled with zeros. If two or more hexwords are specified, they must be separated by one or more blanks.

Yreg hexword1 [hexword2...]
stores the hexadecimal data (hexword1 [hexword2...]) in successive floating-point registers starting at the register specified by reg. The reg operand must be a digit from 0-6. If reg is an odd number, it is adjusted to the preceding even number.

The operands (hexword1 [hexword2...] each represent up to eight hexadecimal digits. If less than eight digits are specified, the string is right justified in a fullword and left-filled with zeros. If two or more hexwords are specified, they must be separated by one or more blanks.

Xreg hexword1 [hexword2...]
stores the hexadecimal data (hexword1 [hexword2...]) in successive control registers starting at the register specified by reg. The reg operand must either be a decimal number from 0-15 or a hexadecimal digit from 0-F. If the virtual machine is in basic control mode, you can store data in register 0 only.

The operands (hexword1 [hexword2...]) each represent up to eight hexadecimal digits. If less than eight digits are specified, the string is right justified in a fullword and left-filled with zeros. If two or more hexwords are specified, they must be separated by one or more blanks.

PSW [hexword1] hexword2
stores the hexadecimal data ([hexword1] hexword2) in the first and second words of the virtual machine's program status word (PSW). If only hexword2 is specified, it is stored into the second word of the PSW. The operands hexword1 and hexword2 must be separated by one or more blanks. They represent up to eight hexadecimal digits. If less than eight digits are specified, the string is right justified and left-filled with zeros.

STATUS stores selected virtual machine data in certain low storage locations of the virtual machine, simulating the hardware store status facility. These locations are permanently assigned locations in real storage. To use the STATUS operand, your virtual machine must be in the Extended Control Mode. The STATUS operand should not be issued for CMS virtual machines or for DOS virtual machines generated for a CPU smaller than a System/360 Model 40. The STATUS operand stores the following data in low storage:

Decimal Address	Hexadecimal Address	Length in Bytes	Data
216	D8	8	CPU Timer
224	E0	8	Clock Comparator
256	100	8	Current PSW
352	160	32	Floating-point registers 0-6
384	180	64	General registers 0-15
448	1C0	64	Control registers 0-15

Response

STORE COMPLETE

SYSTEM

Privilege Class: G

Use the SYSTEM command to simulate the action of the RESET and RESTART buttons on the real computer console, and to clear storage. The RESET function and the CLEAR function leave the virtual machine in a stopped state. An IPL command must be issued after a SYSTEM CLEAR command. After a SYSTEM RESTART, the virtual machine is automatically restarted at the location loaded into the PSW from the doubleword at virtual location zero. The format of the SYSTEM command is:

where:

CLEAR clears virtual storage and virtual storage keys to binary zeros.

RESET clears all pending interrupts and conditions in the virtual machine.

RESTART simulates the hardware system RESTART function by storing the current PSW at virtual location eight and loading, as the new PSW, the doubleword from virtual location zero. Interrupt conditions and storage remain unaffected.

Responses

STORAGE CLEARED - SYSTEM RESET

This response is given if the command SYSTEM CLEAR is entered.

SYSTEM RESET

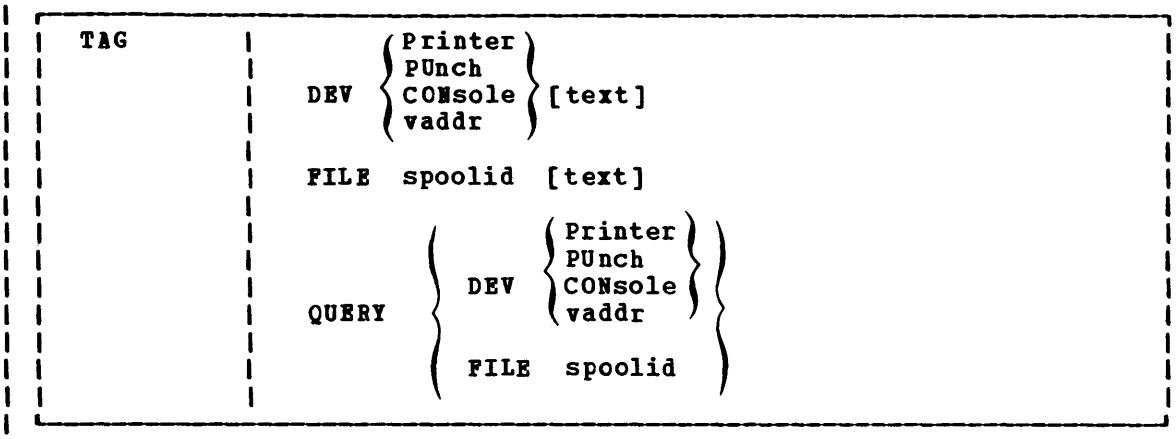
This response is given if the command SYSTEM RESET is entered.

If the command SYSTEM RESTART is entered, no response is given; the virtual machine resumes execution at the address in the virtual PSW loaded from virtual storage location zero.

TAG

Privilege Class: G

Use the TAG command to associate information with a VM/370 spool file, usually for use with a subsystem such as RSCS or a user-written subsystem. The format of the TAG command is:



where:

DEV (PRINTER | PRT | PUNCH | PCH | CONSOLE | vaddr) [text]
: associates information (via the text operand described further on) with your virtual printer, punch, or console, or with the device specified by vaddr.

FILE spoolid [text]
: replaces information previously associated with the file via the text operand with the current text (described further on). This operand can be specified only for reader spool files queued on your virtual machine.

  The spoolid operand is the spool file identification, a number between 1 and 9900 assigned by CP when the spool file was closed.

text
: defines a field (up to 136 characters long) that can contain any information you desire. Typically, this field contains meaningful parameters you want to associate with a spool file. The field of data specified in the TAG text operand is made available to virtual machines using the spool file, but is in no way modified or interpreted by VM/370.

  Certain control and addressing information meaningful to RSCS can be specified in this field. For details on how to use the TAG text operand to transmit files across the RSCS network, refer to the following section, "Using the TAG Text Operand to Transmit Files to Remote Locations."

```
                 ┌      ┌         ┐ ┐
                 │      │ PRINTER │ │
                 │      │ PRT     │ │
                 │ DEV  │ PUNCH   │ │
                 │      │ PCH     │ │
                 │      │ CONSOLE │ │
 QUERY           │      │ vaddr   │ │
                 │      └         ┘ │
                 │ FILE  spoolid    │
                 └                  ┘
```

displays at your terminal the current setting of the TAG text associated with a given spool file or virtual device. The operands used with the TAG QUERY command correspond to the operands used with TAG itself. For example, you create a text setting by issuing the command:

```
TAG DEV PUNCH text
```

To find out the setting of that text field, issue the command:

```
TAG QUERY DEV PUNCH
```

If you know the spoolid of a file queued to your virtual machine reader and you wish to know the text setting for that file, use the command:

```
TAG QUERY FILE spoolid
```

Spool identification (spoolid) numbers can be obtained using the CP QUERY command with the READER, PRINTER or PUNCH operand.

Using the TAG Text Operand to Transmit Files to Remote Locations

The RSCS control program interprets the TAG text operand as addressing and control parameters. If you are spooling a file to the RSCS virtual machine to be transmitted to a remote station, code the TAG text operand as follows:

```
locid [userid] [priority]
```

where:

locid is the location identifier (one to eight alphameric digits) of the location to which the file is being transmitted. Your system programmer can give you the locids of remote stations attached to your virtual machine.

userid is the userid of the VM/370 virtual machine (a one- to eight-character user identification) to which a file is being transmitted. This operand is used by remote stations when they transmit files to the RSCS virtual machine and want the files sent to a particular VM/370 virtual machine. You can ignore this operand if you are not specifying a priority. However, if you are specifying a priority, you must code a userid operand, even though it is ignored by RSCS.

priority is the requested transmission priority, a decimal number between 0 and 99. The highest transmission priority is 0, next highest is 1, and so on. If you wish to specify this operand, you must also specify a userid operand.

Altering Spool File TAG Information

When a spool file you created is closed, it is enqueued on a virtual machine reader of the virtual machine you specified in the SPOOL command. You cannot change the TAG information associated with that file unless the operator of the virtual machine to which the file was spooled (the RSCS operator) transfers the file back to your virtual machine.

To change the TAG information associated with a file in your virtual reader, you can issue the TAG command with the spoolid operand specified:

```
TAG FILE spoolid new text information
```

This command causes previous TAG text information to be completely replaced by the new text specified.

When you enter the command with no new text specified:

```
TAG FILE spoolid
```

the text field associated with the file is set to all blanks.

TERMINAL

Privilege Class: G

Use the TERMINAL command to control the following functions associated with your virtual console:

- Logical line-editing symbols
- Masking of password
- The APL character set
- Signalling of an attention interrupt
- Attention handling mode for your virtual console
- Line length for output on your virtual console

The terminal settings you specify with the TERMINAL command are in effect for only the duration of that terminal session. Whenever you log on, the system defaults are in effect. However, the settings you specify for line-editing and MODE are still in effect when you log on after disconnecting. All the other operands (MASK, APL, ATTN, and LINESIZE) are reset if you log on after disconnecting.

Although you can define line-editing symbols and status with the TERMINAL command, the LINEDIT operand of the SET command determines whether the VM/370 line-editing functions are on or off.

If an error occurs during processing of the command, all functions preceding the one with the error are in effect. The format of the TERMINAL command is:

```
TERMinal  | /  CHardel   { ON   }  \ 1
          | |  LINEDel   { OFF  }  |
          | |  LINENd    { char }  |
          | |  EScape              |
          | |                      |
          | {  Mask      { ON  }   }
          | |  APL       { OFf }   |
          | |  ATtn                |
          | |                      |
          | |  MODE      { CP }    |
          | |            { VM }    |
          | |                      |
          | \  LINESize   nnn      /
```

[1] More than one function can be specified in a single entry of the TERMINAL command. For example:

```
TERMINAL CHARDEL OFF MASK ON LINESIZE 90
```

where:

CHARDEL { ON | OFF | char } defines the logical character delete symbol. If ON is specified, the default symbol becomes the logical character delete symbol. The default symbol is normally @, but depends on what is specified in your VM/370 directory entry. If OFF is specified, no logical character delete symbol is allowed. If char is specified, that character (which must be a special character) becomes the logical character delete symbol. Unless otherwise specified, CHARDEL ON is in effect.

LINEDEL {ON | OFF | char} defines the logical line delete symbol. If ON is specified, the default symbol becomes the logical line delete symbol. The default symbol is normally ¢, but depends on what is specified in your VM/370 directory entry. If OFF is specified, no logical line delete symbol is allowed. If char is specified, that character (which must be a special character) becomes the logical line delete symbol. Unless otherwise specified, LINEDEL ON is in effect.

LINEND {ON | OFF | char} defines the logical line end symbol. If ON is specified, the default symbol becomes the logical line end symbol. The default symbol is normally ¢, but depends on what is specified in your VM/370 directory entry. If OFF is specified, no logical line end symbol is allowed. If char is specified, that character (which must be a special character) becomes the logical line end character. Unless otherwise specified, LINEND ON is in effect.

ESCAPE {ON | OFF | char} defines the logical escape character. If ON is specified, the default symbol becomes the logical escape character. The default symbol is normally ", but depends on what is specified in your VM/370 directory entry. If OFF is specified, no logical escape character is allowed. If char is specified, that character (which must be a special character) becomes the logical escape character. Unless otherwise specified, ESCAPE ON is in effect.

MASK {ON | OFF} controls the typing of a mask line at a typewriter terminal that is not equipped with the Print Inhibit feature, when a password is to be entered. If MASK ON is specified, VM/370 types the mask line. If MASK OFF is specified, the mask line is not typed and it is up to each user to preserve the security of his password. The MASK operand does not apply to the IBM 3215 or to similar system console or display terminals that do not have a Print Inhibit feature. Unless otherwise specified, MASK OFF is in effect.

APL {ON | OFF} controls the use of APL character translation tables. If APL ON is specified, CP uses the translation tables for terminals equipped with the standard APL typing element. If APL OFF is specified, CP uses the normal translation tables (that is, BCD or correspondence code). If APL ON is specified, the LINESIZE value is overridden. Unless otherwise specified, APL OFF is in effect.

This operand cannot be changed for a 3704/3705 device in NCP mode. If however, the terminal is connected to a 2701/2702/2703 line control unit, the operand is valid. 3704/3705 users cannot use this option in NCP mode.

The APL operand is not valid for display type terminals.

Note: APL ON also applies to the 3767 terminal equipped with the APL alternate character selection.

ATTN {ON / OFF} controls signalling of an attention interrupt. If ATTN ON is specified, the exclamation point is displayed when an an attention interrupt occurs. The OFF option suppresses the displaying of the exclamation point (!) and carrier return for those systems that perform special line editing using the Attention key. Unless otherwise specified, ATTN ON is in effect.

The ATTN operand is not valid for display type terminals.

MODE {CP / VM} controls the terminal attention environment. CP specifies that one or more attentions force the virtual machine into the CP environment. VM specifies that one attention is reflected to your virtual machine and that more than one attention forces your virtual machine into the CP environment. VM is the default for all VM/370 users except the primary system operator. For more information see "Interrupting the Execution of a Command" in Section 2.

| LINESIZE nnn specifies the maximum allowable line length for terminal output. nnn can be a number from 1 through 255.

| Note: If APL ON is specified, CP does not separate output lines into LINESIZE segments. Instead, an output length of 1760 is allowed and CP assumes that the APL system has inserted the appropriate carriage control characters.

Responses

None.

TRACE

Privilege Class: G

Use the TRACE command to trace specified virtual machine activity and to record the results at the terminal, on a virtual spooled printer, or on both terminal and printer. If trace output is being recorded at the terminal, the virtual machine stops execution and CP command mode is entered after each output message. This simulates the single cycle function. To resume operation at the virtual machine, the BEGIN command must be entered. If the RUN operand is specified, the virtual machine is not stopped after each output message. If trace output is being recorded on a virtual spooled printer, a CLOSE command must be issued to that printer in order for the trace output to be printed. Successful branches to the next sequential instruction and branch-to-self instructions are not detected by TRACE. Instructions that modify or examine the first two bytes of the next sequential instruction cause erroneous processing for BRANCH and INSTRUCT tracing.

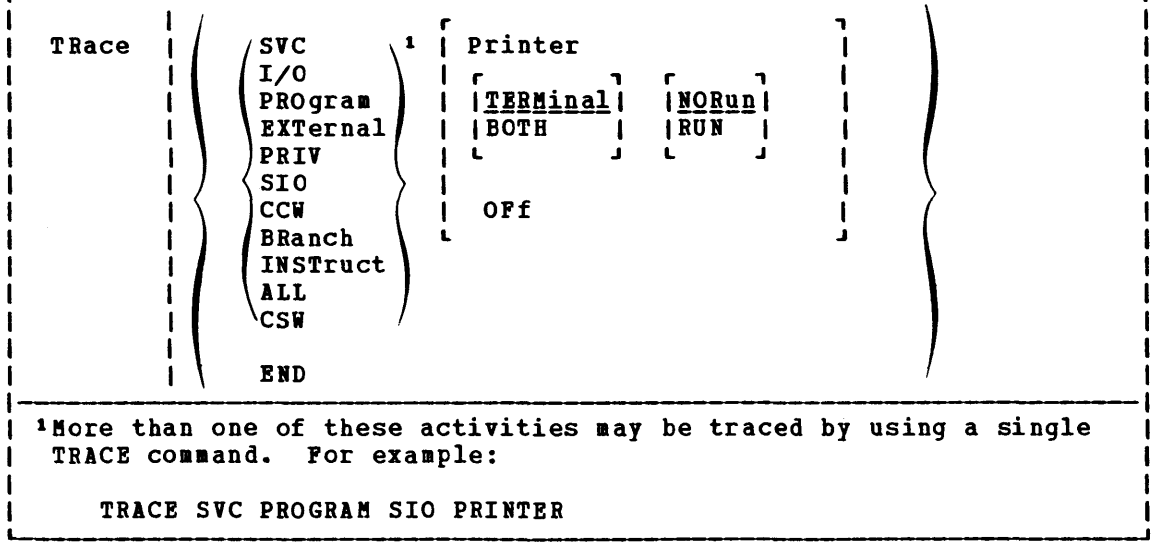
When tracing on a virtual machine with only one printer, the trace data is intermixed with other data sent to the virtual printer. To separate trace information from other data, define another printer with a lower virtual address than the previously defined printer. For example, on a system with 00E defined as the only printer, define a second printer as 00B. The regular output goes to 00E and the trace output goes to 00B.

When operation of a shared system is being traced, the following options cannot be used:

- BRANCH
- INSTRUCT
- ALL

I/O operations for virtual channel-to-channel adapters, with both ends connected to the same virtual machine, cannot be traced.

The format of the TRACE command is:



where:

SVC traces virtual machine SVC interrupts.

I/O traces virtual machine I/O interrupts.

PROGRAM traces virtual machine program interrupts.

EXTERNAL traces virtual machine external interrupts.

PRIV traces all virtual machine non-I/O privileged instructions.

| SIO traces TIO, CLRIO, HIO, HDV and TCH instructions to all virtual devices. Will also trace SIO and SIOF instructions for non-console and non-spool devices only.

CCW traces virtual and real CCWs for non-Spool/non-Console device I/O operations. When CCW tracing is requested, SIO and TIO instructions are also traced.

BRANCH traces all virtual machine interrupts, all PSW instructions, and all successful branches.

INSTRUCT traces all instructions, virtual machine interrupts and successful branches.

ALL traces all instructions, interrupts, successful branches, privilege instructions, and virtual machine I/O operations.

CSW provides contents of virtual and real channel status words at I/O interrupt.

END terminates all tracing activity and prints a termination message.

PRINTER
PRT directs tracing output to a virtual spooled printer.

TERMINAL directs tracing output to the terminal (virtual machine console).

BOTH directs tracing output to both a virtual spooled printer and the terminal.

OFF halts tracing of the specified activities on both the printer and terminal.

NORUN stops program execution after the trace output to the terminal and enters CP command mode.

| Note: If a Diagnose code X'008' is being traced, NORUN has no effect and program execution does not stop.

RUN continues the program execution after the trace output to the terminal has completed and does not enter CP command mode.

Notes:

1. If your virtual machine has the virtual=real option and NOTRANS set on, CP forces CCW translation while tracing either SIO or CCW. When tracing is terminated with the TRACE END command, CCW translation is bypassed again.

2. If the virtual machine assist feature is enabled on your virtual machine, CP turns it off while tracing SVC and program interrupts

(SVC, PRIV, BRANCH, INSTRUCT, or ALL). After the tracing is terminated with the TRACE END command line, CP turns the assist feature on again.

Responses

The following symbols are used in the responses received from TRACE:

Symbol	Meaning
vvvvvv	virtual storage address
tttttt	virtual transfer address or new PSW address
rrrrrr	real storage address
xxxxxxxx	virtual instruction, channel command word, CSW status
yyyyyyyy	real instruction, CCW
ss	argument byte (SSM-byte) for SSM instruction
ns	new system mask after execution of STOSM/STNSM
zz	low order byte of R1 register in an execute instruction (not shown if R1 register is register 0)
zzzzzzzz	referenced data
type	virtual device name (DASD, TAPE, LINE, CONS, RDR, PRT, PUN, GRAF, DEV)
V vadd	virtual device address
R radd	real device address
mnem	mnemonic for instruction
int	interrupt type (SVC, PROG, EXT, I/O)
code	interrupt code number (in hexadecimal)
CC n	condition-code number (0, 1, 2, or 3)
IDAL	Indirect data address list
***	virtual machine interrupt
:::	privileged operations
==>	transfer of control

TRACE STARTED

This response is issued when tracing is initiated.

TRACE ENDED

This response is issued when tracing is suspended.

TCH, TIO, CLRIO, HIO, HDV, SIO, or SIOF

TCH

```
I/O vvvvvv TCH xxxxxxxx type vadd CC n
```

TIO, CLRIO, HIO, or HDV

```
I/O vvvvvv mnem xxxxxxxx type vadd CC n type radd CSW xxxx
```

SIO or SIOF

```
I/O vvvvvv mnem xxxxxxxx type vadd CC n type radd CSW xxxx CAW vvvvvvvv
```

CCW:

```
CCW vvvvvv xxxxxxxx xxxxxxxx rrrrrr yyyyyyyy yyyyyyyy
CCW  IDAL  vvvvvvvv vvvvvvvv  IDAL  00rrrrrr 00rrrrrr
CCW  SEEK  xxxxxxxx xxxxxx    SEEK  yyyyyyyy yyyy
```

The IDAL or SEEK line is included only if applicable. The virtual IDAL is not printed if the real CCW opcode does not match the real CCW.

INSTRUCTION TRACING:

Privileged Instruction:

```
::: vvvvvv SSM   xxxxxxxx ss           (normal SSM)
::: vvvvvv SSM   xxxxxxxx ss  tttttt   (switch to/from translate mode)
::: vvvvvv STOSM xxxxxxxx ns           (normal STOSM)
::: vvvvvv STOSM xxxxxxxx ns  tttttt   (switch to translate mode)
::: vvvvvv STNSM xxxxxxxx ns           (normal STNSM)
::: vvvvvv STNSM xxxxxxxx ns  tttttt   (switch from translate mode)
::: vvvvvv LPSW  xxxxxxxx     tttttttt  tttttttt   (WAIT bit on)
::: vvvvvv LPSW  xxxxxxxx ==> tttttttt  tttttttt   (WAIT bit not on)
::: vvvvvv mnem  xxxxxxxx                          (all others)
```

Executed Instructions:

```
vvvvvv EX xxxxxxxx zz vvvvvv mnem xxxx xxxxxxxx
```

For an executed instruction, where zz (see preceding explanation of symbols) is nonzero, the mnemonic for the executed instruction is given as if the zz byte had been put into the instruction with an OR operation.

All Other Instructions:

```
vvvvvv mnem xxxxxxxx xxxx
```

SUCCESSFUL BRANCH:

```
vvvvvv mnem xxxxxxxx ==> tttttt
```

INTERRUPT (SVC, PROGRAM, or EXTERNAL)

```
*** vvvvvv int code ==> tttttt
```

I/O INTERRUPT (First line given only if "CSW" was specified):

```
CSW V vadd xxxxxxxx xxxxxxxx R radd yyyyyyyy yyyyyyyy
*** vvvvvv I/O vadd ==> tttttt CSW xxxx
```

BRANCH TRACE: (ALL option selected)

Entry for 'branch from' instruction

```
vvvvvv mnem xxxxxxxx     tttttt
```

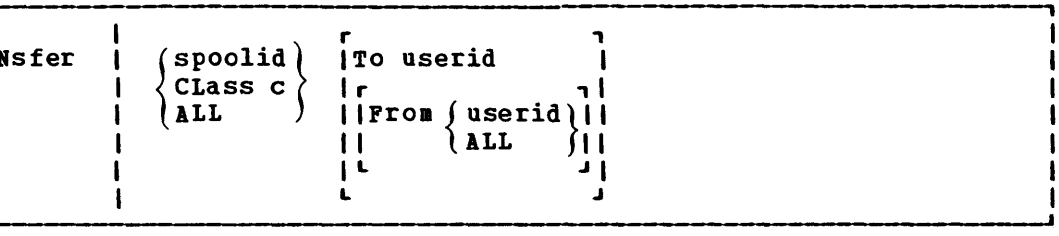
Entry for 'branch to' instruction

```
==> vvvvvv mnem xxxxxxxxxxxx
```

TRANSFER

Privilege Class: G

Use the TRANSFER command to direct your input file to a specified reader or to reclaim virtual reader files that you spooled to another user. The TRANSFER command does not transfer any active spool files. The format of the TRANSFER command is:



where:

spoolid — is the input file to be directed to or retrieved from the named userid.

CLASS c — transfers all input files of the specified class (c). The c is a one-character alphameric field with values from A to Z and from 0 to 9.

ALL — transfers all input spool files.

[TO] userid — is the user to whom the files are to be directed. If the optional keyword TO is omitted, the userid may not be TO or T. The file is deleted from your reader if you use this option.

[FROM] {userid / ALL} — is the user from whom input spool files are to be reclaimed. ALL may be specified to reclaim input spool files that were originated by your virtual machine from all users.

Responses

RDR FILE spoolid TRANSFERRED {TO / FROM} userid

where:

spoolid — is the spool file identification number of the file that is spooled. The number does not change.

TO userid — is the response to the user who currently owns the file and userid is the recipient of the file.

FROM userid — is the response to the user who receives the transferred file and userid is the sender.

{nnnn / NO} FILES TRANSFERRED

This is the response you receive when you issue the TRANSFER command. It is not displayed if you issued the CP SET IMSG OFF command line.

Appendixes

A. Functions of VM/370 Commands

B. Debugging a Problem Program with VM/370

C. Using the CMS Batch Facility

D. CMS Macro Instructions

E. Disk Determination

F. Reserved Filetype Descriptions

Appendix A: Functions of VM/370 Commands

Figures 36 through 44 present a functional summary of commands available in the VM/370 system.

Function	Command	Subcommand or Option	Type
Begin terminal session (identify user to VM/370 system).	LOGON		CP
End terminal session.	LOGOFF		CP
Communicate with other VM/370 users and with the system operator.	MESSAGE		CP
Connect a terminal to a multi-access virtual machine.	DIAL		CP
Disconnect a user's terminal from a virtual machine.	DISCONN		CP
Test terminal hardware.	ECHO		CP
Start or stop console spooling.	SPOOL	CONSOLE	CP
Control terminal input and output.			
• Indicate if accounting data is to be received at the terminal.	SET	ACNT	CP
• Indicate if messages from other users are to be received at the terminal.	SET	MSG	CP
	SET	WNG	CP
• Control output of certain informational responses.	SET	IMSG	CP
• Control line editing functions.	SET	LINEDIT	CP
• Control format of messages received at the terminal.	SET	REDTYPE	CMS
	SET	EMSG	CP
• Set up program function key cataloged procedures.	SET	PFnn	CP
• Print the current screen display on the printer (remote display units only).	SET	PFnn COPY	CP CP
• Get information about terminal control parameters.	QUERY	TERMINAL	CP
	QUERY	PFnn	CP
	QUERY	REDTYPE	CMS
• Set attention handling mode.	TERMINAL	MODE	CP
• Specify method of password entry.	TERMINAL	MASK	CP
• Specify use of additional translation tables.	TERMINAL	APL	CP
• Specify terminal line size.	TERMINAL	LINESIZE	CP
• Specify ATTN key handling procedures.	TERMINAL	ATTN	CP
• Specify that ATTN key will not stop the virtual machine.	SET	RUN	CP
• Specify characters to indicate CPU time interval reporting.	SET	BLIP	CMS
• Specify format of CMS READY message.	SET	RDYMSG	CMS
• Indicate character translations to be done during terminal input and output.	SET	INPUT	CMS
	SET	OUTPUT	CMS

Figure 36. Commands to Control a Terminal Session

Function	Command	Subcommand or Option	Type
Change attributes of a spooled file.	CHANGE		CP
Create a source program file from the terminal.	EDIT		CMS
Invoke the System Assembler to assemble a source program.	ASSEMBLE		CMS
Invoke the VS BASIC Compiler.	VSBASIC		CMS
Invoke the PL/I Optimizing Compiler.	PLIOPT		CMS
Invoke the PL/I Checkout Compiler.	PLIC		CMS
Invoke the PL/I Checkout Compiler and Execute a Program.	PLICR		CMS
Invoke the FORTRAN Code and Go Compiler.	TESTFORT		CMS
Invoke the FORTRAN G Compiler.	FORTGI		CMS
Invoke the FORTRAN H Complier.	FORTHX		CMS
Create or list macro libraries to be used during assemblies or compilations.	MACLIB		CMS
Create or list subroutine libraries.	TXTLIB		CMS
Specify macro libraries to be searched during assemblies or compliations.	GLOBAL	MACLIB	CMS
Specify subroutine libraries to be searched during LOAD and INCLUDE function.	GLOBAL	TXTLIB	CMS
Bring object code into main storage.	LOAD INCLUDE		CMS CMS
Create a MODULE (core-image) file.	GENMOD		CMS
Bring MODULE files into storage.	LOADMOD		CMS
Print a storage map of a MODULE file.	MODMAP		CMS
Build auxiliary module directories.	GENDIRT		CMS
Begin execution of programs which were previously loaded into main storage.	START		CMS

Figure 37. Commands to Develop Programs and Process Data (Part 1 of 2)

Function	Command	Subcommand or Option	Type
Simulate OS Data Definition (DD) JCL cards during program execution.	FILEDEF		CMS
List information about OS data sets or DOS files.	LISTDS		CMS
Load and execute object (TEXT) files.	RUN		CMS
Compile, load, and execute source files.	RUN		CMS
Load and execute core-image (MODULE) files.	RUN		CMS

Figure 37. Commands to Develop Programs and Process Data (Part 2 of 2)

Function	Command	Subcommand or Option	Type
Stop execution at a specified virtual machine location.	ADSTOP		CP
	DEBUG	BREAK	CMS
Resume execution of a stopped virtual machine.	BEGIN		CP
	DEBUG	GO	CMS
Display virtual storage, registers, PSW, and so on.	DISPLAY		CP
	DEBUG	GPR	CMS
	DEBUG	PSW	CMS
	DEBUG	X	CMS
Print the contents of virtual storage locations on the spooled printer.	DUMP		CMS
	DEBUG	DUMP	CMS
Change the contents of registers and storage locations.	STORE		CP
	DEBUG	STORE	CMS
	DEBUG	SET	CMS
Trace virtual machine SVC calls.	SVCTRACE		CMS
Trace virtual machine instructions, I/O operations, SVC calls, and so on.	TRACE		CP
Convert system ABEND dumps to printed output.	VMFDUMP		CMS
Trace events that occur on the real machine.	MONITOR	START CP TRACE	CP
	MONITOR	STOP CP TRACE	CP

Figure 38. Commands to Test and Debug a Program

Function	Command	Subcommand or Option	Type
Create a file from terminal input.	EDIT		CMS
Create a file from card input.	READCARD		CMS
	DISK	LOAD	CMS
Verify the existence of a file on disk.	STATE		CMS
Erase a file (or files) from disk.	ERASE		CMS
	ACCESS	ERASE	CMS
List names of files on disk and their attributes.	LISTFILE		CMS
Display the contents of a file at the terminal.	TYPE		CMS
Print the contents of a file or a member of a library on a spooled printer.	PRINT		CMS
Punch the contents of a disk file on a spooled punch.	PUNCH		CMS
	DISK	DUMP	CMS
Sort the records of a file into ascending order based on specified sort fields.	SORT		CMS
Copy one disk file to another disk file.	MOVEFILE		CMS
	COPYFILE		CMS
Combine several files into one file.	COPYFILE		CMS
Copy data from one device type to another device type.	MOVEFILE		CMS
Change the name of a CMS file.	RENAME		CMS
Compress a file by encoding multiple contiguous occurrences of a single character.	COPYFILE	PACK	CMS
Rearrange the contents of records in a disk file.	COPYFILE		CMS
Perform character translations on specified characters in a disk file.	COPYFILE	TRANS	CMS
Append one file to the end of another file.	COPYFILE	APPEND	CMS
Remove trailing blanks from records in a file.	COPYFILE	TRUNC	CMS
Compare the contents of two disk files.	COMPARE		CMS
Change records in a file based on record sequence numbers.	UPDATE		CMS

Figure 39. Commands to Update Data Files (Part 1 of 2)

Function	Command	Subcommand or Option	Type
Terminate spooling operations on a virtual unit record device or console.	CLOSE		CP
Load the virtual forms control buffer (FCB).	LOADVFCB		CP
Indicate order of processing for spooled files.	ORDER		CP
Remove spooled files from the system.	PURGE		CP
Print documents according to the control words in the document file.	SCRIPT		CMS
Change options in effect for spooling operations.	SPOOL		CP
Spool input files to or from another user.	TRANSFER		CP
Copy data from tape to disk.	TAPE	LOAD	CMS
Copy data from disk to tape.	TAPE	DUMP	CMS
Convert OS partitioned data set (PDS) files or card-image files on tape to format on disk.	TAPPDS		CMS

Figure 39. Commands to Update Data Files (Part 2 of 2)

Function	Command	Subcommand or Option	Type
Logically connect a disk to a virtual machine.	LINK		CP
Logically disconnect a device from a virtual machine.	DETACH		CP
Make files on a disk available to a user.	ACCESS		CMS
Remove accessibility to files.	RELEASE		CMS
Dump a disk to tape.	DDR	DUMP	CMS
Restore a disk from tape.	DDR	RESTORE	CMS
Format disk space in CMS format.	FORMAT		CMS

Figure 40. Commands to Control Disks

Function	Command	Subcommand or Option	Type
Load a virtual machine operating system.	IPL		CP
Alter the virtual machine configuration.	DEFINE		CP
Disconnect the user's terminal from the VM/370 system.	DISCONN		CP
Enter control program commands from a CMS virtual machine.	CP		CMS
Communicate with other virtual machine users or with the system operator.	MSG		CP
Simulate an external interrupt for a virtual machine.	EXTERNAL		CP
Simulate 'not ready' for a virtual device.	NOTREADY		CP
Simulate functions of buttons on the real CPU console.	SYSTEM		CP
Place virtual machine console in dormant state with keyboard locked.	SLEEP		CP
Perform tape rewind action.	REWIND		CP
	TAPE	REW	CMS
Establish VM/370 directory entries.	DIRECT		CMS
Simulate device end interrupt to a virtual machine device.	READY		CP
Simulate console interrupt.	ATTN		CP
	REQUEST		CP
Reset pending interrupts for virtual devices.	RESET		CP
Invoke table of synonyms for CMS command names.	SYNONYM		CMS
Set the functions controlling forms for command operand, options, names.	SET	ABBREV	CMS
		IMPEX	CMS
		IMPCP	CMS

Figure 41. Commands for Virtual Machine Control (Part 1 of 2)

Function	Command	Subcommand or Option	Type
Indicate if I/O is to be done as specified by the virtual machine with no CCW translation by CP.	SET	NOTRANS	CP
Control timer updating.	SET	TIMER	CP
Change or set the number of loader tables for a CMS virtual machine.	SET	LDRTBLS	CMS
Indicate if pages of storage are to be released after the execution of certain CMS commands.	SET	RELPAGE	CMS
Get information about virtual machine status.	QUERY		CP
Set Virtual Machine Assist feature on or off.	SET	ASSIST	CP
Set the pseudo page fault portion of VM/VS Handshaking feature on or off.	SET	PAGEX	CP

Figure 41. Commands for Virtual Machine Control (Part 2 of 2)

Function	Command	Subcommand or Option	Type
Create accounting records for logged on users and reset accounting data.	ACNT		CP
Logically connect or dedicate devices to a virtual machine or to CP.	ATTACH		CP
Logically disconnect devices from a virtual machine or from CP.	DETACH		CP
Disable communication lines.	DISABLE		CP
Enable communication lines.	ENABLE		CP
Force a specific user to log off CP.	FORCE		CP
Terminate active channel CP program on a specific device.	HALT		CP
Control paging activity.	LOCK		CP
	UNLOCK		CP
Request information about real and virtual machine characteristics.	QUERY		CP
Establish system parameters.	SET		CP
Terminate functions and checkpoint system.	SHUTDOWN		CP
Transmit high priority messages to users.	WARNING		CP
Control 3704/3705 control program functions.	NETWORK	LOAD	CP
		DUMP	
		ENABLE	
		DISABLE	
		TRACE	
		HALT	
		SHUTDOWN	
		POLLDLAY	
		QUERY	
		VARY	
		DISPLAY	

Figure 42. Commands to Control VM/370

Function	Command	Subcommand or Option	Type
Restart or reposition the current output of an output spooling device.	BACKSPAC		CP
Change attributes of a closed spool file.	CHANGE		CP
Terminate spooling activity on a virtual spooled unit record device or console.	CLOSE		CP
Halt operations of specified spooling devices following completion of current activity.	DRAIN		CP
Cancel current output on a real unit record device.	FLUSH		CP
Cancel spool HOLD status.	FREE		CP
Defer spooled output of a particular user.	HOLD		CP
Load printer UCS or FCB buffer.	LOADBUF		CP
Cause spooled files to be processed in a specific order.	ORDER		CP
Remove closed spool files from the system.	PURGE		CP
Request information about spool files.	QUERY		CP
Repeat printing or punching of current file on a specific output device.	REPEAT		CP
Force a printer to single space output.	SPACE		CP
Modify spooling options for reader, printer, punch, and console spool files. Initiates and terminates virtual console spooling.	SPOOL		CP
Start spooling device after draining or changing output class.	START		CP
Spool files to or from a user's card reader.	TRANSFER		CP
Direct a file to a remote location.	SPOOL TAG		CP CP
Assign attributes and/or characteristics to a spool file.	TAG SPOOL		CP CP
Request display of the attributes and characteristics associated with a spool file.	TAG		CP

Figure 43. Commands for Spooling Control

Function	Command	Subcommand or Option	Type
Display real storage at terminal.	DCP		CP
Dump real storage to virtual spooled printer.	DMCP		CP
Display conditions that may affect system loading.	INDICATE	LOAD USER QUEUES I/O PAGING	CP
Find locations of control blocks.	LOCATE		CP
Record samples of system data for performance analysis.	MONITOR	DISPLAY ENABLE INTERVAL START STOP	CP
Save virtual machine storage space on disk.	SAVESYS		CP
Perform intensive recording of device activity information.	SET	RECORD	CP
Set the error recording mode for soft machine checks.	SET	MODE	CP
Dump error information which has been recorded by error recording routines.	CPEREP		CMS

Figure 44. Commands for System and Hardware Analysis

How To Start Debugging

Before you can correct any problem, you must recognize that one exists. Next, you must identify the problem, collect information and determine the cause so that the problem can be fixed. When running VM/370, you must also decide whether the problem is in CP, the virtual machine, or the problem program. Once you determine that the problem is in CP or the virtual machine, refer to VM/370: System Programmer's Guide.

A good approach to debugging is:

1. Recognize that a problem exists.
2. Identify the problem type and the area affected.
3. Analyze the data you have available, collect more data if you need it, then isolate the data that pertains to your problem.
4. Finally, determine the cause of the problem and correct it.

DOES A PROBLEM EXIST?

There are four types of problems:

1. Loop
2. Wait state
3. ABEND (Abnormal End)
4. Incorrect results

The most obvious indication of a problem is the abnormal termination of a program. Whenever a program abnormally terminates, a message is issued.

Another obvious indication of a problem is unexpected output. If your output is missing, incorrect, or in a different format than expected, some problem exists.

Unproductive processing time is another symptom of a problem. This problem is not as easily recognized, especially in a time-sharing environment.

IDENTIFYING THE PROBLEM

Two types of problems are easily identified: abnormal termination is indicated by an error message, and unexpected results become apparent once the output is examined. The looping and wait state conditions are not as easily identified.

When using VM/370, you are normally sitting at a terminal and do not have the lights of the CPU control panel to help you. You may have a looping condition if your program takes longer to execute than you anticipated. Also, check your output. If the number of output records or print lines is greater than expected, the output may really be the same information repeated many times. Repetitive output usually indicates a program loop.

Another way to identify a loop is to periodically examine the current PSW. If the PSW instruction address always has the same value, or if the instruction address has a series of repeating values, the program probably is looping.

The wait state is also difficult to recognize when at the terminal. Again, the console lights are unavailable. If your program is taking longer than expected to execute, the virtual machine may be in a wait state. Display the current PSW on the terminal. Periodically, issue the CP command

```
QUERY TIME
```

and compare the elapsed processing time. When the elapsed processing time does not increase, the wait state probably exists.

ANALYZING THE PROBLEM

Once the type of problem is identified, the cause of it must be determined. There are recommended procedures to follow. These procedures are helpful, but they do not identify the cause of the problem in every case. Be resourceful. Use whatever data you have available. If the cause of the problem is not found after the recommended debugging procedures are followed, it may be necessary to undertake the tedious job of desk-checking.

The section, "How To Use VM/370 Facilities To Debug," describes procedures to follow in determining the cause of various problems that can occur in the Control Program, in the virtual machine, or in the problem program. If you determine that there is a problem in CP or the virtual machine operating system, refer to VM/370: System Programmer's Guide for debugging procedures.

How To Use VM/370 Facilities To Debug

Once the problem, and the area where it occurs, are identified, you can gather the information needed to determine the cause of the problem. The type of information you want to look at varies with the type of problem. The tools used to gather the information vary depending upon the area in which the problem occurs. For example, if the problem is looping, you will want to examine the PSW via the CP DISPLAY command.

The following sections describe specific debugging procedures for the various error conditions. The procedures tell you what to do and what debug tool to use. For example, the procedure may say dump storage using the CP DUMP command. The procedure will not tell you how to use the debug tool. Refer to Sections 7 and 8 for a detailed description of each debug tool, including how to invoke it.

PROBLEM PROGRAM ABEND

When an operating system or program does not know how to continue, it abnormally terminates.

If a dump was taken, it was sent to the virtual printer. Issue a CLOSE command to the virtual printer to have the dump print on the real printer.

If the problem can be reproduced, it may be helpful to trace the processing using the CP TRACE command. Also, you can set address stops, and display and alter registers, control words (such as the PSW), and data areas. The CP commands can be very helpful in debugging because you can gather information at various stages in processing. A dump is static and represents the system at only one particular time. Debugging on a virtual machine can often be more flexible than debugging on a real machine.

VM/370 may terminate or reset a virtual machine if a nonrecoverable channel check or machine check occurs in that virtual machine. Hardware errors usually cause this type of virtual machine termination. One of the following messages:

```
DMKMCH616I MACHINE CHECK; USER userid TERMINATED
DMKCCH604I CHANNEL ERROR; DEV xxx; USER userid; MACHINE RESET
```

appears on the CPU console.

The address of the instruction causing the ABEND is found in the problem program area.

Use the PSW and program listing to determine the cause of the ABEND. If the problem is not readily detected, use the CP debugging facilities to monitor the progress and status of the program as it executes. If the program is running under the control of CMS, you can use the CMS debug facilities as well.

UNEXPECTED RESULTS IN A PROBLEM PROGRAM

If a program has inaccurate, missing, or redundant output, a problem exists. Instead of taking dumps of storage and output and doing a lot of desk-checking, you can use the interactive facilities of the virtual machine environment to debug. Using the CP ADSTOP command, you can set an address stop. Set the address stop at a strategic point in the processing, such as before a data transfer or computation. When execution stops, you can display or alter storage and set the next address stop. Also, consider using the CP TRACE command. You can often debug a problem program directly from the terminal.

Both the CP and CMS commands can be used to debug programs under the control of CMS.

PROBLEM PROGRAM DISABLED LOOP

When a disabled loop in a problem program exists, you cannot communicate with the virtual machine's operating system. That means that signalling attention once does not cause an interrupt.

Enter CP mode.

1. Use the CP TRACE command to trace the entire loop. Display general and extended control registers via the CP DISPLAY command.

2. Take a dump via the CP DUMP command.

3. IPL the virtual machine again.

Use the information just gathered, along with listings, to try to find the entry into the loop.

PROBLEM PROGRAM ENABLED LOOP

You should perform the following sequence when attempting to find the cause of an enabled loop:

1. Use the CP TRACE command to trace the entire loop. CMS users can use the CP TRACE command or the CMS SVCTRACE command. Display the PSW and the general registers.

2. Use the CP DUMP command to dump your virtual storage. CMS users can use the debug DUMP subcommand. A standalone dump may be used, but be aware that such a dump destroys the contents of some areas of storage.

3. Consult the source code to search for the faulty instructions, examining previously executed subroutines if necessary. Begin by scanning for instructions that set the condition code or branch on it.

4. If the manner of loop entry is still undetermined, assume that a wild branch has occurred and begin a search for its origin.

PROBLEM PROGRAM DISABLED WAIT

The VM/370 Control Program does not allow the virtual machine to enter a disabled wait state or certain interrupt loops. Instead, CP notifies you of the condition with one of the following messages:

```
DMKDSP450W   CP ENTERED; DISABLED WAIT PSW
DMKDSP451W   CP ENTERED; INVALID PSW
DMKDSP452W   CP ENTERED; EXTERNAL INTERRUPT LOOP
DMKDSP453W   CP ENTERED; PROGRAM INTERRUPT LOOP
```

and enters the CP mode. Use the CP commands to display the following information on the terminal.

- PSW
- CSW
- General registers
- Control registers

Then use the CP DUMP command to take a dump.

If you cannot find the cause of the wait or loop from the information just gathered, try to reproduce the problem, this time tracing the processing via the CP TRACE command.

If CMS is running in the virtual machine, the CMS debugging facilities may also be used to display information, take a dump, or trace the processing. The CMS SVCTRACE and the CP TRACE commands record different information. Figure 45 compares the two.

PROBLEM PROGRAM ENABLED WAIT

If the virtual machine is in an enabled wait state, try to find out why no I/O interrupt has occurred to allow processing to resume.

The Control Program treats one case of an enabled wait in a virtual machine the same as a disabled wait. If the virtual machine does not have the "real timer" option and loads a PSW enabled only for external interrupts, CP issues the message

```
DMKDSP450W CP ENTERED; DISABLED WAIT STATE
```

Since the virtual timer is not decremented while the virtual machine is in a wait state, it cannot cause the external interrupt. A "real timer" runs in both the problem state and wait state and can cause an external interrupt which allows processing to resume.

Comparison of CP and CMS Facilities for Debugging

If you are debugging problems while running CMS, you can choose the CP or CMS debugging tools. Refer to Figure 45 for a comparison of the CP and CMS debugging tools.

Function	CP	CMS
Setting address stops.	Can set only one address stop at a time.	Can set up to 16 address stops at a time.
Dumping contents of storage to the printer.	The dump is printed in hexadecimal format with EBCDIC translation. The storage address of the first byte of each line is identified at the left. The control blocks are formatted.	The dump is printed in hexadecimal format. The storage address of the first byte of each line is identified at the left. The contents of general and floating-point registers are printed at the beginning of the dump.
Displaying the contents of storage and control registers at the terminal.	The display is typed in hexadecimal format with EBCDIC translation. The CP command displays storage keys, floating-point registers and control registers.	The display is typed in hexadecimal format. The CMS commands do not display storage keys, floating-point registers or control registers as the CP command does.
Storing information.	The amount of information stored by the CP command is limited only by the length of the input line. The information can be fullword aligned when stored. CP stores data in floating-point and control registers, as well as in general registers. CP stores data in the PSW, but not in the CAW or CSW. However, data can be stored in the CSW or CAW by specifying the hardware address in the STORE command.	The CMS command stores up to 12 bytes of information. CMS stores data in the general registers but not in the floating-point or control registers. CMS stores data in the PSW, CAW, and CSW.
Tracing information.	CP traces: • All interrupts, instructions, and branches • SVC interrupts • I/O interrupts • Program interrupts • External interrupts • Privileged instructions • All user I/O operations • Virtual and real CCW's • All instructions The CP trace is interactive. You can stop it and display other fields.	CMS traces all SVC interrupts. CMS displays the contents of general and floating-point registers before and after a routine is called. The parameter list is recorded before a routine is called.

Figure 45. Comparison of CP and CMS Facilities for Debugging

The CMS Batch Facility is a VM/370 programming facility that runs under CMS. It allows a VM/370 user to run jobs in batch mode by queuing jobs from either his own virtual machine or the real card reader to a virtual machine dedicated to running batch jobs under the Batch Facility. The Batch Facility machine then executes these jobs, freeing the user's machine for other uses. The accounting routines charge the time used in the batch machine to the originating user.

The Batch Facility virtual machine is generated and controlled at a terminal console under a userid dedicated to execution of jobs in batch mode. The system operator generates a batch machine by performing an IPL of CMS, then entering a command (CMSBATCH), which specifies that the machine is to execute jobs in batch mode. After each job is executed, the Batch Facility reloads itself, thereby providing a continuously running batch machine. Jobs are queued to the batch machine's virtual card reader from either user terminals or the system card reader and executed sequentially. When its virtual reader is empty, the Batch Facility waits for more input.

The Batch Facility is designed for the non-CMS user who requires a system for compiling or executing batch jobs loaded from the real system card reader. The Batch Facility is also useful for the interactive user who has compute-bound jobs such as assemblies and compilations, and for execution of large user programs. This allows interactive users to continue work at their terminals while their time-consuming jobs are run in another virtual machine.

Execution time for CP and CMS commands under the Batch Facility is equivalent to that of commands typed in at a terminal during an interactive CP/CMS session.

Using the Batch Facility Virtual Machine

The Batch Facility is generated on any terminal attached to the VM/370 system. The userid used to run the Batch Facility should be known by all users in the installation.

The interactive users of the Batch Facility must spool their batch jobs to the virtual reader of the batch userid and the non-interactive users must precede the real deck of batch jobs with a CP ID card specifying the userid of the Batch Facility virtual machine. The ID card takes the form:

```
┌──────────────────────────────────────────────────────────────────┐
│    ID   userid                                                   │
└──────────────────────────────────────────────────────────────────┘
```

where ID must begin in card column one and be separated from userid (the Batch Facility virtual machine userid) by one or more blanks.

INPUT TO THE BATCH FACILITY VIRTUAL MACHINE

Any user application or development program written in a language supported by VM/370 may be executed on the Batch Facility virtual machine. However, there are restrictions on programs using certain CP and CMS commands, as described later in this section.

Programs to be executed under the Batch Facility must be delimited by /JOB, and /* control cards. Another control card, the /SET card, may also be used with an input program to set certain limits on the execution of the program.

Input records for the Batch Facility must be in card-image format. Any record not in card-image format is flushed.

The /JOB and /* Cards

The /JOB card identifies the name and userid of a job and provides accounting information for the system. A /JOB card must precede each job to be executed under the Batch Facility. It takes the form:

```
| /JOB  userid accntnum [jobname] [comments]                    |
```

where:

userid — specifies the user identification of the user who sent the job to the batch machine. This userid is charged by the CP accounting routines for the system resources used during a job.

accntnum — is the user's account number. This account number appears in the accounting data that is generated at the end of a user's job. This account number will override the account number in the CP directory entry for that userid.

jobname — is an optional parameter that specifies the name of the job being run.

comments — may be any information the user wishes.

The userid and jobname parameters are used by the Batch Facility for printing of distribution codes for spooled output and for directing job status messages to the interactive user's terminal.

The /* card indicates the end of a job to the Batch Facility. It takes the form:

```
| /*                                                            |
```

The Batch Facility treats all /* cards after the first as null cards. Therefore, if you want to ensure against the previous job not having a /* end-of-job indicator, you should precede your /JOB card with a /* card.

The only exception to the way the Batch Facility handles the /* cards is in the case of the CMS MOVEFILE command. When the input FILEDEF specifies a terminal read, for example FILEDEF INFILE TERM, a /* card must be supplied by the user to delimit that file.

The /SET Card

The /SET card sets limits on a system's time, printing, and punching resources during the execution of a job. It takes the form:

```
/SET [TIME seconds] [PRINT lines] [PUNCH cards]
```

where:

seconds is a decimal value that specifies the maximum number of seconds of virtual CPU time a job can use.

lines is a decimal value that specifies the maximum number of lines a job can print.

cards is a decimal number that specifies the maximum number of cards a job can punch.

The default values for the Batch Facility are set at 32,767 seconds, printed lines, and punched cards per job. Any new limits defined using the /SET card must be less than these maximum settings. The system resources can be set at lesser values than the default values by an installation's system programmer; be sure you know the maximum installation values for batch resource limits before you use the /SET card.

EXEC Files as Input

EXEC files can be used as input to the Batch Facility virtual machine, providing a greater level of programming flexibility for Batch Facility users. The following list shows a few of the uses for EXEC files in manipulating the batch machine. EXEC files can be used:

- To store control statements (/JOB, /*, and /SET) required for execution of queued jobs.
- To substitute keyword values into a program so that the program can be given variable input.
- To skip over portions of programs using conditional EXEC control statements.
- To identify other EXEC files to be used during program execution.
- To integrate many files into a single file for execution under the Batch Facility.

The uses of EXEC files in manipulating the input to the Batch Facility virtual machine are limited only by the programmer's imagination and the rules for writing EXEC files. For rules for writing EXEC files and how they can be used, see the VM/370: EXEC User's Guide.

Restrictions on CP and CMS Commands in Batch Mode

The Batch Facility permits the use of most CP and CMS commands. The exceptions are noted in the following lists.

Only the following CP commands are allowed to control the Batch virtual machine:

CHANGE[1]	MSG
CLOSE[1]	QUERY
DETACH[2]	REWIND
DUMP	SPOOL[1]
DISPLAY	STORE
LINK[3]	TAG

The following CMS commands are disabled under the Batch Facility:

SET
READCARD
DISK LOAD

The FILEDEF command is also disabled for use in defining the virtual card reader.

BATCH FACILITY OUTPUT

The Batch Facility's virtual machine runs with its output spooled to the terminal console and to the printer. If the batch machine's virtual console is connected, CP messages are also printed at the console. The printed copy resulting from execution of a program is printed at the real printer under the submitting userid, with the user's distribution code and a spool file name and type of CMSBATCH JOB (unless a job name was specified on the /JOB card). CMS console output is spooled to a file that is printed following the user's program execution output at the real system printer, with the submitting user's distribution code, a spool filename of BATCH, and a spool filetype of CONSOLE.

Since all the closed printer files are queued for system output under the submitting userid, the submitting user can control (CHANGE, PURGE, or ORDER) these files before processing on the system printer.

At job termination, all spooling devices are closed and all files are released. If the CP TAG command was used to identify spool files or to direct these files to other virtual machines or remote work stations, the Batch Facility resets the spooling devices for the next job. If disk devices were linked to during the job, they are detached by the Batch Facility at job termination time.

The Batch Facility CMS system is then reloaded, reinitializing all nucleus data areas and work areas in storage.

Also at the end of the job, accounting information is punched to the system card punch. This accounting information is in the same format as CP/CMS accounting information.

---

[1]May not be used to affect the virtual card reader.
[2]May not be used to affect spooling devices or the system or IPL disks.
[3]Must be in the following format:
CP LINK userid vaddr vaddr mode password

Error Recovery

When a user job terminates abnormally, the Batch Facility sends an ABEND message in the appropriate userid terminal console, types the message to the BATCH CONSOLE file, and spools a CP dump of the virtual machine (with a heading of userid and jobname) to the printer. The Batch Facility then performs the normal cleanup tasks and starts the next job.

Since the Batch Facility reloads itself (via IPL) after each job, that portion of a job following an ABEND is treated as a new job, which is subsequently flushed.

The macro definitions that are used by the Conversational Monitor System (CMS) are contained in the files CMSLIB MACLIB and OSMACRO MACLIB, which reside on the system disk. CMSLIB MACLIB contains the macros which provide linkages to the CMS function routines, and OSMACRO MACLIB contains Operating System macros which are simulated by CMS. Only the CMS macros are discussed in this section. To obtain a list of the names and locations of macro definitions in a macro library called libname MACLIB, the command MACLIB MAP libname may be issued.

The macros described in this section deal primarily with linkage to the disk and terminal handling routines. WRTERM and RDTERM handle terminal input/output, and FSCB, FSSTATE, FSOPEN, FSREAD, FSWRITE, FSERASE, and FSCLOSE handle I/O to disk. The offline unit record devices may be accessed from an Assembler Language program by using RDCARD, PRINTL, and PUNCHC.

The WRTERM and RDTERM macros each set up a parameter list inline, and issue a CMS supervisor call. For disk I/O, the parameter list is set up in a constant area by the FSCB (File System Control Block) macro, or generated inline if the FSCB option is not specified. The label of that macro is used as a parameter of the FSWRITE, FSREAD, FSOPEN, and FSSTATE macros. The example in Figure 46 shows a typical sequence for writing and reading disk files.

The program example in Figure 46 shows the use of I/O macros. The program copies a file, the "input file," to a new file, the "output file." If the program were processed by GENMOD into a module named BEGIN, it would be invoked with a command line of the form: "BEGIN FILE1 EXEC A1 FILE2 EXEC A1," which would copy FILE1 EXEC (input file) to FILE2 EXEC (output file).

The FSSTATE macro determines the existence of the input file. The error branch will be taken if it does not exist. This is not needed for the output file, since, if it didn't exist, it would be created. If the output file already existed, the new records would be appended to it.

The program then alternates between reading and writing records, until FSREAD returns an error code. Control then goes to EOF, which tests whether end-of-file or some other error occurred. If it was end-of-file, a return code of zero is passed back to the command level in register 15.

If an error occurs, an appropriate response is typed using the WRTERM or LINEDIT macros. Use of the LINEDIT macro at ERR2 and ERR3 causes the value of the return code from FSREAD or FSWRITE to be substituted, in decimal, for the dots in the text of the message to be typed.

Note that, if desired, the FSOPEN and FSCLOSE macros could have been used to open and close the files. It was not necessary to do so in this program because the FSREAD and FSWRITE macros open the file to be read or written. The CMS system automatically closes all files at the end of each command execution.

```
STMT    SOURCE STATEMENT


  1 BEGIN    CSECT
  2          PRINT NOGEN
  3          SAVE  (14,12),,*
  8          BALR  12,0          ESTABLISH ADDRESSABILITY
  9          USING *,12
 10          LA    2,8(,1) R2=ADDR OF INPUT FILEID IN PLIST
 11          LA    3,32(,1)  R3=ADDR OF OUTPUT FILEID IN PLIST
 12 * DETERMINE IF INPUT FILE EXISTS
 13          FSSTATE (2),ERROR=ERR1

 28 * READ A RECORD FROM INPUT FILE AND WRITE ON OUTPUT FILE
 29 RD       FSREAD (2),ERROR=EOF,BUFFER=BUFF1,BSIZE=80
 52          FSWRITE (3),ERROR=ERR2,BUFFER=BUFF1,BSIZE=80
 75          B     RD            LOOP BACK FOR NEXT RECORD

 77 * COME HERE IF ERROR READING INPUT FILE
 78 EOF      C     15,=F'12'     END OF FILE ?
 79          BNE   ERR3          ERROR IF NOT

 81          RETURN (14,12),RC=0
 85 * IF INPUT FILE DOES NOT EXIST
 86 ERR1     WRTERM 'FILE NOT FOUND',EDIT=YES
 96          B     ERRET

 98 * IF ERROR WRITING FILE
 99 ERR2     LR    10,15         SAVE RET CODE IN REG 10
100          LINEDIT TEXT='ERROR CODE ..... IN WRITING FILE',SUB=(DEC,(10))
115          B     ERRET

117 * IF READING ERROR WAS NOT NORMAL END OF FILE
118 ERR3     LR    10,15         SAVE RET CODE IN REG 10
119          LINEDIT TEXT='ERROR CODE ..... IN READING FILE',SUB=(DEC,(10))

135 ERRET    RETURN (14,12),RC=1 RETURN TO CALLER

141 BUFF1    DS    CL80
142          END
143               =CL8'RDBUF'
144               =CL8'WRBUF'
145               =A(BUFF1)
146               =F'80'
147               =F'12'
```

Figure 46. A Sample Listing of a Program that Uses CMS Macros

COMPSWT Macro

PURPOSE:

The COMPSWT macro causes the COMPSWT flag in the OSSFLAGS byte of the Nucleus Constant Area to be turned on or off.

FORMAT:

```
| [label] | COMPSWT | ON|OFF |
```

USAGE:

The compiler switch flag COMPSWT in the Nucleus Constant Area determines whether the OS macros LINK, LOAD, XCTL, and ATTACH will use the CMS INCLUDE command or the CMS LOADMOD command. With COMPSWT off, the program called by LINK, LOAD, XCTL, or ATTACH must be a relocatable object module residing in a file with the filetype TEXT or TXTLIB. It is called via the INCLUDE command. With COMPSWT on, the program called by LINK, LOAD, XCTL, or ATTACH must be an absolute core-image module in a file with the filetype MODULE. It is called via the LOADMOD command.

FSCB Macro

PURPOSE:

The FSCB macro creates a File System Control Block, which is used as the parameter list to the file system routines.

FORMAT:

```
┌──────────────────────────────────────────────────────────┐
| [label] | FSCB | [fileid] [,RECFM=format]              |
|         |      | [,BUFFER=buffer] [,BSIZE=size]        |
|         |      | [,RECNO=number] [,NOREC=numrec]       |
└──────────────────────────────────────────────────────────┘
```

label — is an optional statement label.

fileid — specifies the file identifier enclosed in quote marks and separated by blanks ('filename filetype filemode'). If filemode is omitted, "A1" is assumed.

format — F indicates fixed-length records (default).
V indicates the variable-length records.

buffer — specifies the address of the I/O buffer.

size — specifies the size of the buffer in bytes.

number — specifies the relative record number of the next record to be accessed (default = 0).

numrec — specifies the number of records to be read (default = 1).

Note: The above options must be specified as self-defining terms.

USAGE:

The FSCB macro generates a File System Control Block for the specified file. It may be used as the parameter list to the file system routines.

ERROR CONDITIONS:

Errors in macro operand coding will cause Assembler errors.

FSCLOSE Macro

PURPOSE:

The FSCLOSE macro causes an open file to be closed and its current status saved on the disk.

FORMAT:

```
[label] FSCLOSE [fileid] [,FSCB=fscb] [,ERROR=erraddr]
```

label is an optional statement label.

fileid specifies the file identifier. 'fileid' may be coded even if FSCB= is also specified. It may be:

'fn ft fm'	fileid enclosed in quote marks and separated by blanks. If fm is omitted, "A1" is assumed.
(reg)	a register from 2 to 15 containing the address of the fileid (18 characters).

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

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

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

USAGE:

FSCLOSE should be used within EXEC procedures to close files that are no longer needed for processing. The FSCLOSE macro causes the specified file to be closed. Either a fileid or FSCB must be specified.

ERROR CONDITIONS:

If an error occurs, processing terminates, and control is passed to erraddr (if one was provided) or back to the calling program's next sequential instruction. Register 15 contains the following error code:

Code	Meaning
6	File not open

FSERASE Macro

PURPOSE:

The FSERASE macro causes one or more files to be deleted from the user's disk.

FORMAT:

```
|[label]| FSERASE |[fileid][,FSCB=fscb][,ERROR=erraddr] |
```

label is an optional statement label.

fileid specifies the file identifier. 'fileid' may be coded even if FSCB= is also specified. It may be:

'fn ft fm'	fileid enclosed in quote 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 specifies the address of an FSCB. It may be:

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

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

USAGE:

The FSERASE macro causes the specified file to be deleted from the user's disk. Either a fileid or FSCB must be specified.

ERROR CONDITIONS:

If an error occurs, register 1 points to the FSCB for the file having the error and register 15 contains one of the following error codes:

Code	Meaning
24	Parameter list error
28	File not found

FSOPEN Macro

PURPOSE:

The FSOPEN macro causes the file to be made ready for either input or output.

FORMAT:

```
┌──────────────────────────────────────────────────────────┐
| [label] | FSOPEN | [fileid] [,FSCB=fscb] [,options] |
|         |        |   [,ERROR=erraddr]                   |
└──────────────────────────────────────────────────────────┘
```

label is an optional statement label.

fileid specifies the file identifier. 'fileid' may be coded even if FSCB= is also specified. It may be:

'fn ft fm'	the fileid enclosed in quote 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 specifies the address of an FSCB. It may be:

label	the label on an FSCB macro.
(reg)	a register containing the address of an FSCB. If FSCB= is not coded, an fscb is created inline, pointed to by register 1, upon return from the macro call.

options any of the options of the FSCB macro instruction may be specified here. Only the BUFFER, RECNO, NOREC, and BSIZE (variable file only) options have any effect for an already existing file. These options may be specified as either a self-defining term or a register enclosed in parentheses.

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

USAGE:

The FSOPEN macro causes the FSCB to be filled in from the file status table for an already existing file. A file status table entry is created for a new file. Either a fileid or FSCB must be specified. On return, register 1 points to the FSCB for the file.

FSOPEN is used mainly to determine the existence of the file about to be processed.

ERROR CONDITIONS:

If an error occurs, processing terminates, and control is passed to erraddr (if one was provided) or back to the calling program's next sequential instruction. Register 15 contains one of the following error codes:

Code	Meaning
28	File does not exist
20	Invalid file identifier

FSREAD Macro

PURPOSE:

The FSREAD macro causes a record to be read from disk.

FORMAT:

```
┌──────────────────────────────────────────────────────────┐
| [label] | FSREAD | [fileid] [,FSCB=fscb] [,options]      |
|         |        |  [,ERROR=erraddr]                     |
└──────────────────────────────────────────────────────────┘
```

label is an optional statement label.

fileid specifies the file identifier. 'fileid' may be coded even if FSCB= is also specified. It may be:

'fn ft fm'	the fileid enclosed in quote 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 specifies the address of an FSCB. It may be:

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

options the BUFFER, NOREC, BSIZE, and RECNO options of the FSCB macro instruction may be specified here. If the FSCB parameter is omitted, the BUFFER and BSIZE options must be specified. These options may be specified as either a self-defining term or a register enclosed in parentheses.

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

USAGE:

The FSREAD macro reads one record into the user's buffer. If the RECNO option is specified, that specified record is read. Otherwise, the next record in the file is read. If RECNO is specified, it must be reset to zero by the next FSREAD in order to read the file sequentially from the originally specified record number. Either a fileid or FSCB must be specified. On return, register 0 contains the number of bytes read. Register 1 points to the FSCB.

ERROR CONDITIONS:

If an error occurs, processing terminates, and control is passed to erraddr (if one was provided) or back to the calling program's next sequential instruction. Register 15 contains one of the following error codes:

Code	Meaning
1	File not found
2	Invalid buffer address
3	Permanent I/O error

Code	Meaning
5	Number of records is less than or equal to zero, or greater than 32768
7	Invalid record format (only checked for at the first read after finis or ipl)
8	Incorrect length
9	File open for output
11	Number of records greater than 1 for variable length file
12	End of file
13	Variable length file has invalid displacement in active file table
14	Invalid character in filename
15	Invalid character in filetype

FSSTATE Macro

PURPOSE:

The FSSTATE macro returns to the user the current status of the specified file.

FORMAT:

```
|[label]| FSSTATE | [fileid] [,FSCB=fscb] [,ERROR=erraddr] |
```

label is an optional statement label.

fileid specifies the file identifier. 'fileid' may be coded even if FSCB= is also specified. It may be:

'fn ft fm'	the fileid enclosed in quote 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 specifies the address of an FSCB. It may be:

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

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

USAGE:

The FSSTATE macro causes the existence of the specified file to be verified. Register 1 points to a copy of the File Status Table (FST) for the specified file. Either a fileid or FSCB must be specified.

ERROR CONDITIONS:

If an error occurs, processing terminates, and control is passed to erraddr (if one was provided) or back to the calling program's next sequential instruction. 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 Macro

PURPOSE:

The FSWRITE macro causes a record to be written to the disk.

FORMAT:

```
| [label] | FSWRITE | [fileid] [,FSCB=fscb] [,options]  |
|         |         | [,ERROR=erraddr]                  |
```

label is an optional statement label.

fileid specifies the file identifier. 'fileid' may be coded even if FSCB= is also specified. It may be:

- 'fn ft fm' the fileid enclosed in quote 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 specifies the address of an FSCB. It may be:

- label the label on an FSCB macro.
- (reg) a register containing the address of an FSCB. If FSCB= is not coded, BUFFER= and BSIZE= must be coded.

options any of the options of the FSCB macro instruction may be specified here. Only the BUFFER, RECNO, and BSIZE options have any effect on an already existing file. These options may be specified as either a self-defining term or a register enclosed in parentheses.

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

USAGE:

The FSWRITE macro writes one record onto the user's disk. If the RECNO option is specified, that specified record is written. Otherwise, the next sequential record in the file is written.

To write a file beginning with a specific record in the file, two FSWRITE statements must be issued: the first to specify the record number where the writing is to begin, for example, RECNO=5 begins the write operation at record number 5 in the specified file. The second FSWRITE statement must set the RECNO option to zero, which will cause the remaining records in the file to be written sequentially. Either a fileid or FSCB must be specified. On return, register 1 points to the FSCB for the file.

Using the BSIZE option of the FSCB macro, you can write multiple records in a single FSWRITE statement. For example, if you wish to write 20 records at once, you code BSIZE=1600 (the length of the record times the number of records). Variable length records cannot be written like this, however. They must be written sequentially each with its own length specified.

ERROR CONDITIONS:

If an error occurs, processing terminates, and control is passed to erraddr (if one was provided) or back to the calling program's next sequential instruction. Register 15 contains one of the following error codes:

Code	Meaning
2	Invalid buffer address
4	First character of filemode is illegal
5	Second character of filemode is illegal
6	Item number too large
7	Attempt to skip over unwritten variable length item
8	Buffer size not specified
9	File open for input
10	Maximum number files reached
11	Record format not F or V
12	Attempt to write on read-only disk
13	Disk is full
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)
20	Invalid character detected in filename
21	Invalid character detected in filetype

HNDEXT Macro

PURPOSE:

The HNDEXT macro causes external interrupts to be passed to the user's routine for processing.

FORMAT:

```
| [label] | HNDEXT | function[,address] |
```

label is an optional statement label.

function specifies whether the interrupts are to be trapped (SET) or cleared (CLR).

address specifies the address of the routine to process the interrupts. This parameter is required only when the SET function is specified.

USAGE:

The HNDEXT macro causes CMS to pass control to the user's routine when an external interrupt occurs. When the user is given control, all virtual interrupts except multiplexer are disabled. On entry to the user's routine, register 1 points to a save area used for saving the registers and PSW at the time of the external interrupt, starting at label GRS, as shown below. Register 13 points to an 18-fullword save area starting at label UAREA, shown below. Register 14 contains the address the user's routine must return to when this processing is complete, and register 15 contains the address of the user's processing routine. After HNDEXT CLR is issued, an external interrupt other than a timer interrupt will cause DEBUG to be entered.

SAVE AREA FORMAT:

	Displacement	
Label	Dec	Hex
GRS	0	0
FRS	64	40
PSW	96	60
UAREA	108	68
END	180	B4

ERROR CONDITIONS:

None.

HNDINT Macro

PURPOSE:

The HNDINT macro causes interrupts for the specified devices to be passed to the user's routine for processing.

FORMAT:

```
| [label] | HNDINT | function,(devname[,address,devaddr,when]) ... |
|         |        | [,ERROR=erraddr]                               |
```

label is an optional statement label.

function specifies whether the interrupts are to be trapped (SET) or cleared (CLR).

devname specifies a four character symbolic name for the device.

address specifies the address of the routine to process the interrupts. An address of zero indicates interrupts for the device are to be ignored.

devaddr specifies the device address, in hexadecimal, of the device whose interrupts are to be trapped.

when specifies that the interrupts are to be reflected immediately (ASAP) or only after the WAITD macro is issued (WAIT).

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

Note: address, devaddr, and when need not be specified if CLR is specified.

USAGE:

The HNDINT macro causes CMS to pass control to the user's routine when an interrupt is received from one of the specified devices. The user routine will be entered immediately if ASAP is specified or a WAITD macro has been issued for the device. If neither condition exists, the interrupt will be stacked, until a WAITD macro is issued for the device. When the user is given control, all I/O interrupts and external interrupts are disabled. On entry to the user's routine, registers 0 and 1 contain the I/O old PSW, registers 2 and 3 contain the CSW, register 4 contains the interrupting device address, register 14 contains the address that the user's routine must return to when his processing is complete, and register 15 contains the address of the user's processing routine. When processing is complete, the user must return to CMS via register 14 and indicate in register 15 whether processing is complete. A zero in register 15 indicates that the user has completed handling the interrupt and a nonzero register 15 indicates that another interrupt is expected. The CLR function indicates that the user is finished processiing interrupts for this device. For additional information, see the documentation for WAITD.

ERROR CONDITIONS:

If an error is detected, register 15 contains a 1, indicating that the devaddr or address is invalid.

HNDSVC Macro

PURPOSE:

The HNDSVC macro causes interrupts for the specified supervisor call (SVC) numbers to be passed to the user's routine for processing.

FORMAT:

```
| [label] | HNDSVC | {set|clr},{(svcnum,address)|svcnum}  |
|         |        |  [,{(svcnum,address)|svcnum}]...     |
|         |        |   [,ERROR=erraddr]                   |
```

label — is an optional statement label.

svcnum — specifies a number from 0 to 200 or 206 to 255 that corresponds to the SVC number to be handled by the user routine. (SVC's 204 and 205 are reserved for future use.)

address — specifies the address of the routine to process the interrupt. This field need not be specified with the CLR function.

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

USAGE:

The HNDSVC macro causes CMS to pass control to the user's routine when one of the specified SVC instructions is issued. On entry to the user's routine the following conditions exist:

Register No.	Contents
0-11 and 15	As they were when the SVC was executed.
12	Address of user SVC handler routine.
13	Address of an 18-fullword save area for use by the user's SVC handler routine.
14	Return address in CMS.

The user must return to CMS via register 14 when he has completed processing the interrupt. The CLR function indicates that the user is finished handling interrupts for the specified SVC number.

ERROR CONDITIONS:

If an error occurs, processing terminates, and control is passed to erraddr (if one was provided) or back to the calling program's next sequential instruction. 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 Macro

PURPOSE:

The LINEDIT macro provides a method for performing conversions into EBCDIC, and, optionally, typing the results at the terminal.

FORMAT:

```
| [label] |  LINEDIT  | [MF=I|L|(E,addr)|(E,(reg))]
|         |           | [,TEXT='message-text']
|         |           | [,TEXTA=address|(reg)]
|         |           | [,COMP=YES|NO]
|         |           | [,DOT=YES|NO]
|         |           | [,SUB=(type,value[,type,value]..)]
|         |           | [,DISP=TYPE|NONE|SIO|PRINT|CPCOMM]
|         |           | [,BUFFA=address|(reg)]
|         |           | [,RENT=YES|NO ]
|         |           | [,MAXSUBS=number]
```

where the option list consists of the following keyword parameters (described in greater detail in the section on "LINEDIT Macro Parameters"):

MF=I|L|(E,addr)|(E,(reg)) to specify macro form

TEXT='message-text' to specify message text

TEXTA=address|(reg) to specify message text address

COMP=YES|NO to suppress multiple blank compression

DOT=YES|NO to suppress final period on message text

SUB=(type,value[,type,value]...) to specify a substitution list where:

```
type = HEX
     = HEXA
     = DEC
     = DECA
     = HEX4A
     = CHARA
     = CHAR8A

value= expression
     = (reg)
     = address
     = (address,length) length being equal either
          to number or (reg)
```

DISP=TYPE|NONE|SIO|PRINT|CPCOMM to specify the action to be taken with message text.

BUFFA=address|(reg) to specify return buffer address

RENT=yes|no to specify whether reentrant code must be generated.

MAXSUBS=number to specify the maximum number of substitutions with MF=L.

Note: Registers 1 and 15 should not be used as argument registers to the LINEDIT macro; they are used as work registers by the macro.

LINEDIT MACRO PARAMETERS

MF Parameter

The MF parameter specifies whether the form of the macro is to be standard, list, or execute. This parameter is coded as follows:

```
MF=I|L|(E,addr)|(E,(reg))
```

with the following meanings:

MF=I (Standard form)

This is the most usual form, and the default. It generates an inline PLIST, and generates a call to the LINEDIT routine so that the message will be typed at the terminal.

The standard form will generate only reentrant code, and may be used under all circumstances except the following:

- If more than one substitution is made.
- If TEXTA=(reg) is used. Note that TEXTA=address can be used in the standard form, with the address referenced in the PLIST by an address constant.
- If the BUFFA parameter is used.

MF=(E,addr) or MF=(E,(reg)) (Execute form)

If used in this form, the macro will generate code to construct a PLIST at the specified address, rather than using an inline PLIST. The code will then call LINEDIT, passing the address of the newly constructed PLIST.

The address of the construction area for the PLIST is given by "addr," to specify an address which can be obtained by a Load Address (LA) instruction, or by "(reg)," where the register specified points to the area.

In either case, the specified area must be large enough to construct a PLIST. The size of a PLIST is variable, depending upon the number of substitutions made, and whether BUFFA was specified. The list form of the LINEDIT macro is used to reserve space for the PLIST.

MF=L (List form)

This form generates a PLIST only. It does not generate any executable code. It is used to reserve space for a PLIST to be constructed using the MF=(E,....) execute form. The size of the area reserved depends upon the number of substitutions to be made, as specified with the MAXSUBS parameter.

For example,

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

reserves space for a PLIST which may hold up to five substitution parameters.

TEXT Parameter

This parameter specifies the message text as a character string between single quotes. Single quotes appearing in the message text should be coded as two single quotes. This parameter is coded as follows:

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

If the TEXT Parameter is coded, the TEXTA parameter must not be coded.

TEXTA Parameter

This parameter is an alternative to the TEXT parameter. It specifies the address of the message text. The message text consists of a one-byte length field, followed by the actual text of the message. The parameter is coded as follows:

```
TEXTA=address|(reg)
```

If used in the standard or list form of the macro, the "address" is used in an address constant. The "(reg)" form then may not be used. If used in the execute form, the "address" is obtained by a Load Address (LA) instruction. If the "(reg)" form is used, then the specified register (other than registers 1 and 15) contains the address of the message text area. If the TEXTA parameter is coded, the TEXT parameter must not be coded.

COMP Parameter

This parameter specifies how multiple blanks in the message are to be handled. With COMP=YES, the default, the LINEDIT routine compresses all multiple blanks in the message into a single blank. If COMP=NO is coded, this compression is not done.

DOT Parameter

The DOT parameter indicates whether the end of the message will have a period. With DOT=YES, the default, the LINEDIT routine places a period (dot) at the end of the message. If DOT=NO is coded, this is not done.

SUB Parameter

This parameter is used to specify a substitution list that allows a message to have a value substituted into it at execution time. Whenever the LINEDIT scanning routine discovers two or more consecutive periods in the message text, it assumes that a substitution is to be made.

The number of consecutive periods indicates the length (or maximum length) of the substituted field, after conversion, if any. The SUB parameter is coded as follows:

```
SUB=(type,value[,type,value]...)
```

where each "type,value" pair specifies the type and value of the substitution. Possible "type,value" substitutions are:

HEX,(reg)
The value in the specified register (other than registers 1 and 15) is to be converted to graphic hexadecimal form. If fewer than eight consecutive periods are coded in the message text, then leading digits will be truncated. Leading zeros are not automatically truncated.

HEX,expression
The given expression, which will be evaluated by means of a Load Address (LA) instruction, is converted as indicated above into graphic hexadecimal form.

HEXA,address
The "address" specifies a fullword that is to be converted to graphic hexadecimal, as above.

HEXA,(reg)
The register (other than registers 1 and 15) points to a fullword that is to be converted to graphic hexadecimal form, as above.

DEC,(reg)
The value in the specified register (other than registers 1 and 15) is to be converted to graphic decimal form and inserted into the message text. Leading zeros will be suppressed. If the number is negative, then a leading minus sign will be inserted.

DEC,expression
The given expression, which will be evaluated by means of a Load Address (LA) instruction, is converted as indicated above into graphic decimal.

DECA,address
The "address" specifies a fullword which is to be converted to graphic decimal, as above.

DECA,(reg)
The specified register (other than registers 1 and 15) contains the address of a fullword that is to be converted to graphic decimal, as above.

HEX4A,address or HEX4A,(reg)
Is used for graphic hexadeciaml dumps of other than fullwords. The "address" (or a register other than 1 or 15) points to the first byte to be converted to graphic hexadecimal form. A blank is inserted into the message text after each four characters of input (eight characters of output) have been coverted. Thus, if an 8-byte input field (16 hexadecimal digits) is to be converted, 17 periods should be coded. The output field length is determined by the number of periods in the message text. The length, in bytes, of the field to be converted may be specified as follows:

```
HEX4A,(address|(reg),length|(reg))
```

CHARA,address or CHARA,(reg) The "address" (or a register other than 1 or 15) points to the first byte of a field that is to be substituted as a character string. The output field length is determined by the number of consecutive periods in the message text. The length, in bytes, of the field to be substituted may be specified as follows:

```
CHARA,(address|(reg),length|(reg))
```

CHAR8A,address or CHAR8A,(reg) Is useful if a string of PLIST arguments (for example, a fileid) is to be substituted into the message text. LINEDIT will insert a blank after each 8 characters of the substitution string. Thus, to substitute an 18-byte fileid, code 20 periods in the message text, to allow for the inserted blanks. The "address" (or a register other than 1 or 15) points to the first byte of the input substitution string. The length, in bytes, of the input string may be specified as follows:

```
CHAR8A,(address|(reg),length|(reg))
```

Note that registers 1 and 15 may not be used for the address or length specification.

DISP Parameter

This option specifies what LINEDIT is to do with the message which it has created. Specifications for the DISP parameter are:

DISP=TYPE (the default) specifies that the message is to be typed on the terminal using TYPLIN.

DISP=SIO specifies that the message is to be typed at the terminal using SIO. This option is necessary if, for example, free storage pointers are destroyed. No CONWAIT function is performed.

DISP=NONE specifies that no output whatsoever is to occur. This option is useful with the BUFFA parameter, described below.

DISP=PRINT specifies that the line is to be printed on the virtual printer.

DISP=CPCOMM specifies that the line is to be passed to CP to be executed as a CP console command.

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

The first ten characters of the line are the "code" and the rest of the line is the "text" of an error message. In a standard VM/370 error message, the code of an error message is in the following format:

```
mmmmmmnnnt
```

where mmmmmm is the name of the module issuing the message, nnn is the message number, and t is the message type. With LINEDIT, you can code error messages for your own programs. You need not follow the VM/370 conventions for the first nine characters of the code, but the tenth character must specify a recognized VM/370 message type if error message editing is to occur. The recognized types are:

Message Type	Tenth Character of Message Line
INFORMATION	I
WARNING	W
ERROR	E

If one of these letters is the tenth character in the line, the line will be typed in accordance with the current EMSG conditions that are controlled by the CP SET EMSG command. If the tenth character is anything else, the entire line will be typed as a regular line, with no distinction made between code and text.

BUFFA Parameter

This parameter specifies the address of a buffer. It is coded as follows:

```
BUFFA=address|(reg)
```

If a register is specified, it must not be register 1 or 15.

If this option is specified, then, in addition to the action specified by the DISP parameter, the message text is copied into the buffer at the specified address. The length of the text is inserted into the first byte of the buffer, and the message text into subsequent bytes.

RENT Parameter

The "standard" form of the LINEDIT macro generates reentrant code unless one of the following is true:

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

If reentrant code is not generated, and RENT=YES (the default) is in effect, the LINEDIT macro expansion contains an MNOTE statement warning the user that nonreentrant code is being generated. If the user does not object to nonreentrant code, and does not wish to have the MNOTE appear, he should code RENT=NO. The RENT=NO coding merely suppresses the MNOTE statement; it has no effect on the expansion of the LINEDIT macro.

MAXSUBS Parameter

This parameter is used only in the list form (MF=L) of the macro, which is used to reserve space. It specifies the maximum number of substitutions to be made, so that enough space for those substitutions can be reserved. It is coded:

```
MAXSUBS=number
```

LINEDIT CODING EXAMPLES

The following examples illustrate some options of the LINEDIT macro. Because of the limitations of space, some of the longer coding examples appear on two lines. However, normal coding conventions for continuation statements should be used in actual coding.

Example 1

The simplest use of the LINEDIT macro is to type a message without any substitutions whatsover.

```
LINEDIT TEXT='THIS    IS    A    LINE'
```

causes the following to be typed out on the terminal when it is executed:

```
THIS IS A LINE.
```

Note that multiple blanks are removed from the message, and a period is placed at the end of the line.

Example 2

The real power of the LINEDIT macro is in the capability of making substitutions. Whenever the LINEDIT processor finds two or more consecutive periods in the message text, it makes a substitution from the 'SUB' list provided in the macro call.

To illustrate the simplest type of substitution, using the DEC option, suppose the following is executed:

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

If the Assembler language program contains a statement of the form "VALUE EQU 2003", then when this macro is executed, the line,

```
THE VALUE IS 2008.
```

is typed at the terminal.

Example 3

Usually some value computed during execution of the program is requested. To type out the contents of a register, the following can be used:

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

If, when this macro is executed, register 3 contains the decimal value 10345, then the following line is typed out:

```
VALUE = 10345.
```

Example 4

The DECA option is used to specify the address of a fullword in storage which contains the value to be typed out. The address of evaluated by means of a Load Address (LA) instruction. Execution of:

```
LINEDIT TEXT='VALUE = ......',SUB=(DECA,LOC)
```

would cause the value in the fullword at location LOC to be typed out in decimal format.

Similarly, the macro call

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

would cause the value in the fullword whose address is in register 3 to be typed out in decimal.

Note: As the examples given show, when the keyword (such as DEC) does not end in an 'A', then a value is being specified as the next argument. When the keyword (such as DECA) does end in 'A', then the address of a value is being specified. When the next argument is enclosed in parentheses, the value or address is to be found in the register, other than 1 or 15, that was specified.

Example 5

The number of consecutive periods appearing in the message text is usually significant. In DEC and DECA, it indicates the maximum number of characters (including the minus sign) which will appear in the number. If the number is too large to fit in the field, then high-order digits will be truncated.

To be certain that high-order digits will never be truncated, code ten periods in the message text for DEC and DECA, as in:

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

Example 6

The HEX option is similar in use to the DEC option, except that the substituted values are typed out in hexadecimal rather than in decimal. Execution of:

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

would cause the value of LOC to be substituted in hexadecimal format, so that the following might be typed out:

```
'LOC' IS LOCATED AT 0201AC.
```

Note that the leading zeros are not removed.

Example 7

Parentheses are used to type out the current value of a register in hexadecimal. For example, if register 3 contained the value C0031FC8, then the macro

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

would cause the line,

```
VALUE = FC8.
```

to be typed out. Notice that since three periods were specified, just the last three hexadecimal digits of the register contents were typed out.

Example 8

The HEXA option is used if the address of a fullword in storage containing the desired value is specified. Execution of:

```
LINEDIT TEXT='HEX VAL = .....',SUB=(HEXA,VAL)
```

will cause the last five hexadecimal digits of the fullword at VAL to be substituted into the message text.

And if

```
LINEDIT TEXT='HEX VAL = ......',SUB(HEXA,(5))
```

is executed, then the last six hexadecimal digits of the fullword whose address is in register 5 will be substituted.

Example 9

The HEX4A option is used for dumping a variable number of bytes of storage on the terminal. The dump has the format:

```
hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh ....
```

where each "h" stands for a hexadecimal digit. The number of periods in the message text specifies the size of the output field and, hence, indirectly specifies the number of bytes of "input" data to be converted.

Each byte of input data is converted into two hexadecimal digits for substitution. After each four bytes of input (eight bytes of output), a blank is inserted. Thus, if a dump of seven bytes of data in this format is requested, 15 periods must be coded. The extra period is for the blank to be inserted. The macro,

```
LINEDIT TEXT='VALUES ARE ...............',
      SUB=(HEX4A,(6))
```

would cause a line to be displayed as follows:

```
VALUES ARE 0A23F115 78ACFE.
```

where the values being dumped start at the address that is specified in register 6.

Example 10

The CHARA option is used to make a straight character substitution. The number of periods indicates the number of characters to be substituted:

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

would cause the seven characters starting at location LOC to be substituted into the text, to produce a message such as:

```
NAME IS 'LUCINDA'.
```

Multiple blanks in the substituted string will be removed.

Example 11

The CHAR8A option is similar to the CHARA option, except that a blank is inserted into the output text after each group of eight characters. This is particularly useful for dumping PLISTs, since a blank is inserted after each doubleword in the PLIST. For example,

```
LINEDIT TEXT='FILE ..................... NOT
      FOUND',SUB=(CHAR8A,PLIST+8)
```

could cause the following output:

```
FILE X ASSEMBLE A1 NOT FOUND.
```

Note that 20 periods were coded, to include space for the two inserted blanks. In the final edited line, multiple blanks are reduced to a single blank. If the CHARA option had been used, with 18 periods specified, then the same message would have appeared as:

```
FILE X ASSEMBLEA1 NOT FOUND.
```

Example 12

To make several substitutions in a given line, it is necessary only to code several groups of periods in the message text, and several pairs of substitution values in the SUB parameter list.

The following macro:

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

would generate a line like the following:

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

Each group of periods represents one substitution, and each pair of SUB arguments represents one value to be substituted.

Example 13

To reserve space in a work area for a LINEDIT PLIST, the "list" form of the LINEDIT macro (MF=L) must be used. Specifying

```
LINLIST LINEDIT MF=L,MAXSUBS=6
```

will reserve space for a LINEDIT PLIST, and give that space the label 'LINLIST'. The parameter 'MAXSUBS=6' indicates that enough space is to be reserved so that a PLIST with space for six or fewer substitution pairs can be built.

If a larger MAXSUBS value is specified, then a larger amount of space is reserved for PLIST contruction.

Example 14

The execute form of the LINEDIT macro does not generate a PLIST as part of its expansion, as does the standard form. (Examples 1 through 12 use the standard form of the macro.) Instead, the execute form expands to create a PLIST in a work area. The space for the PLIST should have been reserved using MF=L as indicated in Example 13. To call LINEDIT after creating a PLIST in the work area reserved by the LINEDIT macro in the last example, the following form could be used:

```
LINEDIT MF=(E,LINLIST),other options ...
```

The "other options" may be any of the options discussed in preceding examples, including TEXT and SUB parameters.

Example 15

Specifying substitution length -- 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 exact size of the input data area. Sometimes it is desirable to determine at execution time the size of the input data area.

These sizes can be specified with the CHARA, CHAR8A, and HEX4A conversion types.

The following substitution list might appear in a LINEDIT macro:

```
SUB=(CHARA,LOC,HEX4A,(3))
```

When specified in this manner, the lengths of the input data fields are determined by the number of periods in the message text. To specify the length of the input data field, the following alternative form may be used:

```
SUB=(CHARA,(LOC,(4)),HEX4A,((3),(5)))
```

In the latter format, where each substitution pair has the form "type,(address,length)", register 4 is specified as containing the length of the data area at location LOC, and register 5 is specified as containing the length of the data area pointed to by register 3.

Example 16

Specifying a length is particularly useful in the special case where only one character is to be changed in a CHARA substitution. To do this, two periods must be coded into the message, and a length of one must be specified, as in the example:

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

The single letter (length=1) at location PLIST+24 is substituted into the field consisting of two periods.

Note: No matter what input length is specified, the final substituted result cannot be longer than the number of periods in the message text. If the input field length and the output length (specified by the number of consecutive periods) conflict, the output length will govern, and the input length will be truncated accordingly.

PRINTL Macro

PURPOSE:

The PRINTL macro causes a line to be written to a virtual printer.

FORMAT:

```
| [label] | PRINTL | line [,length] [,ERROR=erraddr] |
```

label is an optional statement label.

line specifies the line to be printed. It may be in one of three forms:

'linetext' text enclosed in quote 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. If omitted, 133 is assumed unless the text is enclosed in quotes. In this case, the number of characters between the quotes is used. The length may be specified in two ways:

(reg) a register containing the length.

n a self-defining term indicating the length.

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

USAGE:

The PRINTL macro causes the specified line to be written to the printer. The maximum length allowed is 151 characters if printing on a virtual 3211 or 133 characters if printing on a virtual 1403. The first character in the line is used as the carriage control character. It must be ASCII control character code or a machine carriage control code other than X'C1' or X'C3'. Since X'C1' and X'C3' are valid for both ASCII control character and machine carriage control code, if coded, they are interpreted as ASCII control character codes. If a valid control character is not specified, a write command with a carriage control character to space one line after printing is used.

The valid ASCII control characters are as follows:

Character	Hex Code	Meaning
⌀	40	Space 1 line before printing
0	F0	Space 2 lines before printing
-	60	Space 3 lines before printing
+	4E	Suppress space before printing

Character	Hex Code	Meaning
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

ERROR CONDITIONS:

If an error occurs, processing terminates, and control is passed to erraddr (if one was provided) or back to the calling program's next sequential instruction. Register 15 contains one of the following error codes:

Code	Meaning
1	Line too long.
2	Channel 12 punch sensed (virtual 3211 only).
3	Channel 9 punch sensed (virtual 3211 only).
4	Intervention required.
5	Unknown error.
100	Printer not attached.

Note: If the error was a result of sensing a channel 9 or 12 punch while writing to a virtual 3211 using ASA control characters, the operation terminated after the carriage movement but before writing the line. In order to avoid an extra blank line, the print buffer should be rewritten using the 01 channel command code (write without spacing).

PUNCHC Macro

PURPOSE:

The PUNCHC macro causes a line to be written to a virtual card punch.

FORMAT:

```
| [label] | PUNCHC| line [,ERROR=erraddr] |
```

label is an optional statement label.

line specifies the line to be punched. It may be in one of three forms:

'linetext' text enclosed in quote marks.

lineaddr the symbolic address of the line.

(reg) a register containing the address of the line.

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

USAGE:

The PUNCHC macro causes the specified line to be written to the card punch. No stacker selecting is allowed. The line length must be 80 characters.

ERROR CONDITIONS:

If an error occurs, processing terminates, and control is passed to erraddr (if one was provided) or back to the calling program's next sequential instruction. Register 15 contains one of the following error codes:

Code	Meaning
2	Unit check.
3	Unknown error.
100	Punch not attached.

RDCARD Macro

PURPOSE:

The RDCARD macro causes a line to be read from a virtual card reader.

FORMAT:

```
| [label] | RDCARD | buffer [, length] [, ERROR = erraddr] |
```

label — is an optional statement label.

buffer — specifies the buffer address into which the card is to be read. It may be either of two forms:

- bufaddr — the symbolic address of the buffer.
- (reg) — a register containing the address of the buffer.

length — specifies the length of the 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.

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

USAGE:

The RDCARD macro causes a line to be read from the card reader. No stacker selecting is allowed. On return register 0 contains the length of the card read.

ERROR CONDITIONS:

If an error occurs, processing terminates, and control is passed to erraddr (if one was provided) or back to the calling program's next sequential instruction. 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 Macro

PURPOSE:

The RDTAPE macro causes a record to be read from the specified tape drive.

FORMAT:

```
| [label] | RDTAPE | buffer,length [,device] [,MODE=mode] |
|         |        |  [,ERROR=erradr]                      |
```

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 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 is assumed. It may be specified in either of two forms:

TAPn n indicates the symbolic tape number.

cuu indicates the virtual device address.

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 or 1600 for a 9-track tape.

trtch indicates the tape recording technique for 7-track tape. One of the following must be specified:

0 - 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.

Note: Mode need not be specified for a read operation to a 9-track tape or to an 800 BPI odd parity 7-track tape with the Data Converter and the Translator off.

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

USAGE:

The RDTAPE macro causes a record to be read from the specified tape drive. On return, register 0 and the eighth word of the parameter list contain the number of bytes read.

ERROR CONDITIONS:

If an error occurs, processing terminates, and control is passed to erraddr (if one was provided) or back to the calling program's next sequential instruction. 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 not attached.

RDTERM Macro

PURPOSE:

The RDTERM macro causes a line to be read from the user's terminal.

FORMAT:

```
┌─────────┬─────────┬──────────────────────────────────────────────┬──────────┬─────┐
│         │         │                                              │          │ ┌─┐ │
│ [label] │ RDTERM  │ buffer [,EDIT=code] [LENGTH=length]          │ATTREST=  │ │YES│ │
│         │         │                                              │          │ │NO │ │
└─────────┴─────────┴──────────────────────────────────────────────┴──────────┴─┘─┘
```

label is an optional statement label.

buffer specifies the address of a 130-character buffer into which the line is to be read. It may be either of two forms:

- lineaddr the symbolic address of the buffer.
- (reg) a register containing the address of the buffer.

code specifies the type of editing, if any, to be performed on the input line. If this is not coded, YES is assumed.

- 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 upper case.
- YES indicates PAD+UPCASE.
- PHYS indicates that a physical line is to be read. When PHYS is specified, the LENGTH and ATTREST operands may also be entered.

length specifies the length of the caller's buffer area. If not specified, 130 is assumed. When EDIT=PHYS, as many as 2030 bytes may be read.

ATTREST= specifies the handling of an attention signal during a read operation. The normal handling (ATTREST=YES) is to restart the operation. If EDIT=PHYS and an explicit LENGTH is given, NO may be specified. In these cases, an attention signal is treated as a normal end of the operation.

USAGE:

The RDTERM macro causes a line to be read from the user's terminal. On return, register 0 contains the number of characters read.

ERROR CONDITIONS:

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

Code	Meaning
2	An invalid parameter was given.
4	The read was terminated by an attention signal. Possible only when ATTREST=NO

REGEQU Macro

PURPOSE:

The REGEQU macro equates symbolic names to the general-purpose, floating-point, and extended-control registers.

FORMAT:

```
|       | REGEQU |                          |
```

USAGE:

The REGEQU macro causes the following equate statements to be generated. The symbolic names may be used as register specifications in assembly language statements. The register usage will then appear in the cross reference listing.

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 Macro

PURPOSE:

The TAPECTL macro causes the specified tape to be positioned according to the specified function code.

FORMAT:

```
|[label]|TAPECTL|function [,device][,MODE=mode][,ERROR=erraddr]|
```

label is an optional statement label.

function specifies the control function to be performed. It must be one of the following codes:

Code	Action
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 is assumed. It may be in either of two forms:

TAPn — n indicates the symbolic tape number.

cuu — indicates the virtual device address.

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 or 1600 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.

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

USAGE:

The TAPECTL macro causes the control operation to be performed on the specified tape drive.

ERROR CONDITIONS:

If an error occurs, processing terminates, and control is passed to erraddr (if one was provided) or back to the calling program's next sequential instruction. 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.

WAITD Macro

PURPOSE:

The WAITD macro causes the program to wait until the next interrupt occurs for the specified device.

FORMAT:

```
| [label] | WAITD | device...[,devicen] [,Error=erraddr] |
```

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 the device number.

- user a name of up to four characters specified in a previous HNDINT macro.

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

USAGE:

The WAITD macro causes the program to wait for an interrupt from one of the specified devices. It should be issued to ensure completion of an I/O operation. If an interrupt has been received and not processed from a device specified in the WAITD macro (the when parameter in the HNDINT macro had to be WAIT) the interrupt for the device will be entered. When the interrupt has been completely processed, control will be returned to the caller with the name of the interrupting device in Register 1. If an interrupt has been received and processed from a device specified in the WAITD macro (the when parameter in the HNDINT macro has to be ASAP) the wait condition is considered satisfied and return to the caller is made as previously described.

An interrupt received from a device specified ASAP in an HNDINT macro will be handled. Processing this interrupt will satisfy a WAITD issued later for that device.

The interrupt routine determines if an interrupt is considered "processed" or if more interrupts are necessary to satisfy the wait condition. For additional information see the HNDINT macro documentation.

ERROR CONDITIONS:

When an error is detected, register 15 contains a 1 to indicate that an invalid device number was specified.

WAITT Macro

PURPOSE:

The WAITT macro causes the program to wait until all of the pending I/O operations to the user's terminal are completed.

FORMAT:

```
| [label] | WAITT |
```

label is an optional statement label.

USAGE:

The WAITT macro is used to synchronize terminal I/O operations.

ERROR CONDITIONS:

None.

WRTAPE Macro

PURPOSE:

The WRTAPE macro causes a record to be written on the specified tape drive.

FORMAT:

```
| [label] | WRTAPE | buffer,length [,device]          |
|         |        | [,MODE=mode] [,ERROR=ERRADDR] |
```

label is an optional statement label.

buffer specifies the address of the record to be written. It may be in either of two forms:

- 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 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 is assumed. It may be in either of two forms:

- TAPn — n indicates the symbolic tape number.
- cuu — indicates the virtual device address.

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 or 1600 for a 9-track tape.
- trtch indicates the tape recording technique for 7-track tape. One of the following must be specified:
  - 0 - 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.

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

USAGE:

The WRTAPE macro causes a record to be written to the specified tape drive.

ERROR CONDITIONS:

If an error occurs, processing terminates, and control is passed to erraddr (if one was provided) or back to the calling program's next sequential instruction. Register 15 contains one of the following error codes:

Code	Meaning
1	Invalid function or parameter list.
2	End-of-tape or End-of-file.
3	Permanent I/O error.
4	Illegal device id.
5	Tape not attached.
6	Tape is file protected.

WRTERM Macro

PURPOSE:

The WRTERM macro causes a line to be displayed at the user's terminal.

FORMAT:

```
┌──────────────────────────────────────────────────┐
| [label] | WRTERM | line [,length] [,EDIT=code ] |
|         |        |   [,COLOR=color]             |
└──────────────────────────────────────────────────┘
```

label is an optional statement label.

line specifies the line to be displayed. It may be one of three forms:

'linetext' text enclosed in quote 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 quote marks in the macro, the length parameter 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.

code specifies whether the line is to be edited:

YES indicates that trailing blanks are to be removed and a carriage return added to the end of the line.

NC indicates that trailing blanks are nct to be removed and no carriage return is to be added.

LONG indicates the line may exceed 130 bytes, and is to be transmitted from the caller's buffer area. No editing is performed.

color indicates in which color the line is to be typed if the typewriter terminal has a two-color ribbon:

B indicates that the line is to be typed in black.

R indicates that the line is to be typed in red.

USAGE:

The WRTERM macro causes the specified line to be displayed at the user's terminal. The maximum line length is 130 characters for a black line and 126 characters for a red line. If EDIT=LONG, COLOR must be given as 'B'. In this case, as many as 1760 bytes may be sent with a single WRTERM macro. The caller is responsible for properly embedding terminal control characters in the data, and for the integrity of the data from issuance of WRTERM until the data has been sent. The WAITT macro may be used to ensure that I/O is complete before the buffer is modified.

ERROR CONDITIONS:

None.

Figure 47 relates CMS commands, method of specifying filemode, and the criteria used in choosing a disk directory for reading and writing. Symbols used in the table are:

Symbol	Meaning
command	CMS command name
fm	Explicit mode letter can be specified
=	Write disk to Read disk
*	Refer to all disks in a set search order
d	Default mode: let system determine the mode
-	Null mode; unable to specify mode letter in this command
Reading	The criteria for choosing the disk from which to read
N/A	Not applicable, command does not cause any reading to be done
fm	Read from the specified disk
*R	Refer to all disks in the standard search order
1R	Read only from the primary disk
*cuu	All occurrences of the address
Writing	Indicates criteria for selecting the disk to write onto
N/A	Not applicable, command does not cause any writing to be done
fm	Write onto the specified disk
R	Write onto the disk from which a file was read (or its parent)
*W	Choose any read/write disk in the standard search pattern
1W	Attempt to write onto the primary disk
cuu	Write to the specified address
*WS	First read/write disk with enough space
*1	First disk where file is found if disk is in read/write status

Command	Filemode	Reading	Writing
ACCESS	mode	fm	N/A
	d	1R	N/A
ASM3705	-	*R	R,1W,*W
ASSEMBLE	-	*R	R,1W,*W
BASIC	-	*R	R,1W
COBOL[1]	-	*R	R,1W,*W
COMPARE	fm	fm	N/A
	*	*R	N/A
CONVERT[1]	fm	fm	fm
COPYFILE	fm	fm	fm
	=	N/A	R
	*	*R	N/A
CP	-	N/A	N/A
DEBUG	-	N/A	N/A
DIRECT			
DISK DUMP	fm	fm	N/A
	*	*R	N/A
	d	1R	N/A
DISK LOAD	-	N/A	1W
EDIT	fm	fm	fm
	*	*R	R
	d	1R	R
ERASE	fm	N/A	fm
	*	N/A	*W
	d	N/A	1W
EXEC	-	*R	N/A
FILEDEF	fm	fm	fm
	d	1R	1W
	*	*R	N/A
FORMAT	mode	N/A	fm
FORTGI[1]	-	*R	R,*W
FORTHX[1]	-	*R	R,*W
FORTRAN[1]	-	*R	R,*W
GEN3705	-	*R	1W
GENDIRT	-	N/A	N/A

[1]IBM Program Products

Figure 47. Disk Determination (Part 1 of 3)

Command	Filemode	Reading	Writing
GENMOD	fm	N/A	fm
	*	N/A	1W
	d	N/A	1W
GLOBAL	–	N/A	N/A
GOFORT[1]	–	*R	R,*W
INCLUDE	–	*R	1W
LISTDS			
LISTFILE	fm	fm	1W
	*	*R	1W
	d	1R	1W
LKED	–	*R	R,1W,*W
LOAD	–	*R	1W
LOADMOD	–	*R	N/A
	fm	fm	N/A
MACLIB	–	*R	R,1W
MODMAP	–	*R	N/A
MOVEFILE	–	N/A	N/A
NCPDUMP	–	*R	1W
PLIC[1]	–	*R	R,1W,*W
PLICR[1]	–	*R	R,1W,*W
PLIOPT[1]	–	*R	R,1W,*W
PRINT	fm	fm	N/A
	d	1R	N/A
	*	*R	N/A
PUNCH	fm	fm	N/A
	d	1R	N/A
	*	*R	N/A
QUERY	–	N/A	N/A
READCARD	fm	N/A	fm
	d	N/A	1W
	*	N/A	1W
RELEASE	–	*cuu	cuu
	mode	fm	fm

[1]IBM Program Products

Figure 47. Disk Determination (Part 2 of 3)

Command	Filemode	Reading	Writing
RENAME	fm	fm	fm
	*	*R	N/A
	=	N/A	R
RUN	fm	fm	N/A
	*	*R	N/A
	d	1R	N/A
SAVENCP	–	*R	N/A
SCRIPT[2]	–	*R	1W
SET	–	N/A	N/A
SORT	fm	fm	fm
	*	*R	R,1W
START	–	N/A	1W
STATE	fm	N/A	N/A
	*	N/A	N/A
	d	N/A	N/A
SVCTRACE	–	N/A	N/A
SYNONYM	fm	fm	N/A
	*	*R	N/A
	d	1R	N/A
TAPE DUMP	fm	fm	N/A
	*	*R	N/A
	d	1R	N/A
TAPE LOAD	fm	N/A	fm
	d	N/A	1W
TAPE SCAN	–	N/A	N/A
TAPE SKIP	–	N/A	N/A
TAPPDS	fm	N/A	fm
	d	N/A	1W
TESTCOB[1]	–	*R	R,1W,*W
TESTFORT[1]	–	*R	*L,*WS
TXTLIB	–	*R	R,1W
TYPE	fm	fm	N/A
	*	*R	N/A
	d	1R	N/A
UPDATE	–	*R	R,1W
	fm	fm	fm,R,1W
	d	1R	1W
	*	*R	R,1W
VSBASIC[1]	–	*R	R,1W
ZAP	–	*R	R

[1]IBM Program Products
[2]IBM User Installed Program

Figure 47. Disk Determination (Part 3 of 3)

Figure 48 shows filetypes that have special uses in CMS.

Filetype	Command	Usage	Filename	Format		Contents
				RECFM	LRECL	
ASSEMBLE	ASSEMBLE	input	fn	F	80	Assembler language source statements
ASM3705	ASM3705	input	fn	F	80	3704/3705 assembler
	GEN3705	output	fn(nn)	F	80	source statements
AUXxxxx	UPDATE	input	fn	F	80	Auxiliary update file
BASIC	BASIC	input	fn	F	≤256	BASIC language source statements
BASDATA	BASIC execution	execution-time files	fn	U	≤3440	User input and output files
CMSUT1	READCARD	inter-	READ	F	80	
	COPYFILE	mediate	COPYFILE			
	DISK	work	DISKLOAD			
	TAPE	file	TAPELOAD			
	UPDATE		fn			
	INCLUDE		DMSLDR			
	LOAD		DMSLDR			
	MACLIB		DMSLBM			
CNTRL	UPDATE	input	fn	F	80	Control file update
COBOL	COBOL[1]	input	fn	F	80	COBOL source statements
COPY	MACLIB	input	fn	F	80	COPY control cards and macro definitions
DIRECT	DIRECT	input	fn	F	80	User Directory entries
EXEC	EXEC	input	fn	F	80	EXEC statements
	LISTFILE	output	CMS			
	GEN3705	output	fn	F	80	
FREEFORT	GOFORT[1]	input	fn	V	≤81	FREEFORM FORTRAN source statements
FORTRAN	FORTGI[1]	input	fn	F	80	FORTRAN source statements
	FORTHX[1]					
	GOFORT[1]					
	TESTFORT[1]					
FTnnF001	FORTRAN execution	input/output	fn			User input and output files

[1]IBM Program Products

Figure 48. Reserved Filetypes (Part 1 of 3)

Filetype	Command	Usage	Filename	Format		Contents
				RECFM	LRECL	
LISTING	ASSEMBLE	output	fn	F	121	Processor printed output
	ASM3705	output	fn			
	GOFORT[1]					
	FORTGI[1]					
	FORTHX[1]					
	COBOL[1]	output	fn			
	PLIC[1]					
	PLICR[1]					
	PLIOPT[1]	output	fn			
	TESTCOB[1]	input	fn	F	121	COBOL processor output used as input to SOURCE subcommand of TESTCOB
LKEDIT	LKED	output	fn	F	121	Listing
LOADLIB	LKED	output	fn	V	≤260	3704/3705 control program load modules
	ZAP	input	fn	V	≤260	
MACLIB	GLOBAL	library	fn	Library contains dictionary and members		Macro definitions
	MACLIB	MACLIB	fn			Macro definitions
MACRO	MACLIB	input	fn	F	80	Macro definitions
MAP	INCLUDE	output	LOAD			Module map
	LOAD	output	LOAD			Module map
	MACLIB	output	fn			Library map
	TXTLIB	output	fn			Library map
MEMO			fn	F	80	
MODULE	GENMOD	output	fn	V		Non-relocatable object file
	LOADMOD	input	fn			
	MODMAP	input	fn			
PLI or PLIOPT	PLIOPT[1]	input	fn	F		PL/I source statements
	PLIC	input	fn			
	PLICR	input	fn			
SCRIPT	SCRIPT[2]	input	fn	V	≤133	Input to SCRIPT processor
SYNONYM	SYNONYM	reference	fn	F	80	Command name synonyms
SYSUT1,2,3	ASM3705	work	fn			
	ASSEMBLE	work	fn	V		
	COBOL[1]	work	fn			
	LKED	work	fn			
	PLIOPT[1]	work	fn			

[1]IBM Program Products
[2]IBM Installed User Program

Figure 48. Reserved Filetypes (Part 2 of 3)

Filetype	Command	Usage	Filename	Format		Contents
				RECFM	LRECL	
SYSUT4	COBOL[1]	work	fn	F	80	
	LKED					
	PLIC					
	PLICR					
	TESTCOB[1]	input			512	Used as input to TESTCOB
TESTFORT	TESTFORT[1]	output	fn	VB	125	Processor printed output
TEXT	ASSEMBLE	output	fn	F	80	Object code
	ASM3705	output	fn	F	80	3704/3705 source code and job control language statements
	COBOL[1]	output	fn			Object code
	GEN3705	output	fn(Ln)	F	80	Linkage editor control statements for 3704/3705 control programs
	INCLUDE	input	fn			Object code
	LKED	input	fn			Object code and LKED control cards
	LOAD	input	fn			Object code
	PLIOPT[1]	output	fn			Object code
	TXTLIB	input	fn			Object code
	GOFORT[1]	output	fn			Object file
	FORTGI[1]	output				
	FORTHX[1]	output				
	TESTFORT[1]	input				
TXTLIB	GLOBAL	library	fn	Library contains dictionary and members		Object decks
	TXTLIB	output	fn			
UPDATE	UPDATE	input	fn	F	80	UPDATE control cards
UPDLOG	UPDATE	output	fn	F		UPDATE log
UPDATE	UPDATE	input	fn	F	80	Local updates
VSBASIC	VSBASIC	input	fn	F	≤256	VSBASIC language source statements
VSBDATA	VSBDATA	execution time files	fn	V	≤140	VSBASIC user input/output files

[1]IBM Program Products

Figure 48. Reserved Filetypes (Part 3 of 3)

READER'S COMMENTS

Title: IBM Virtual Machine Facility/370: Command Language Guide for General Users

Order No. GC20-1804-3

Please check or fill in the items; adding explanations/comments in the space provided.

Which of the following terms best describes your job?

- ☐ Customer Engineer
- ☐ Engineer
- ☐ Instructor
- ☐ Manager
- ☐ Mathematician
- ☐ Operator
- ☐ Programmer
- ☐ Sales Representative
- ☐ Student/Trainee
- ☐ Systems Analyst
- ☐ Systems Engineer
- ☐ Other (explain below)

How did you use this publication?

☐ Introductory text ☐ Reference manual ☐ Student/ ☐ Instructor text

☐ Other (explain) _____

Did you find the material easy to read and understand? ☐ Yes ☐ No (explain below)

Did you find the material organized for convenient use? ☐ Yes ☐ No (explain below)

Specific criticisms (explain below)

Clarifications on pages _____

Additions on pages _____

Deletions on pages _____

Errors on pages _____

Explanations and other comments:

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

Trim Along This Line

GC20-1804-3

YOUR COMMENTS PLEASE . . .

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance and/or additional publications or to suggest programming changes will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.* Your comments will be carefully reviewed by the person or persons responsible for writing and publishing this material. All comments or suggestions become the property of IBM.

FOLD FOLD

FIRST CLASS
PERMIT NO. 172
BURLINGTON, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
VM/370 PUBLICATIONS
24 NEW ENGLAND EXECUTIVE PARK
BURLINGTON, MASS. 01803

FOLD FOLD

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

Trim Along This Line

IBM VM/370: Command Lang Guide for Genl Users Printed in U.S.A. GC20-1804-3