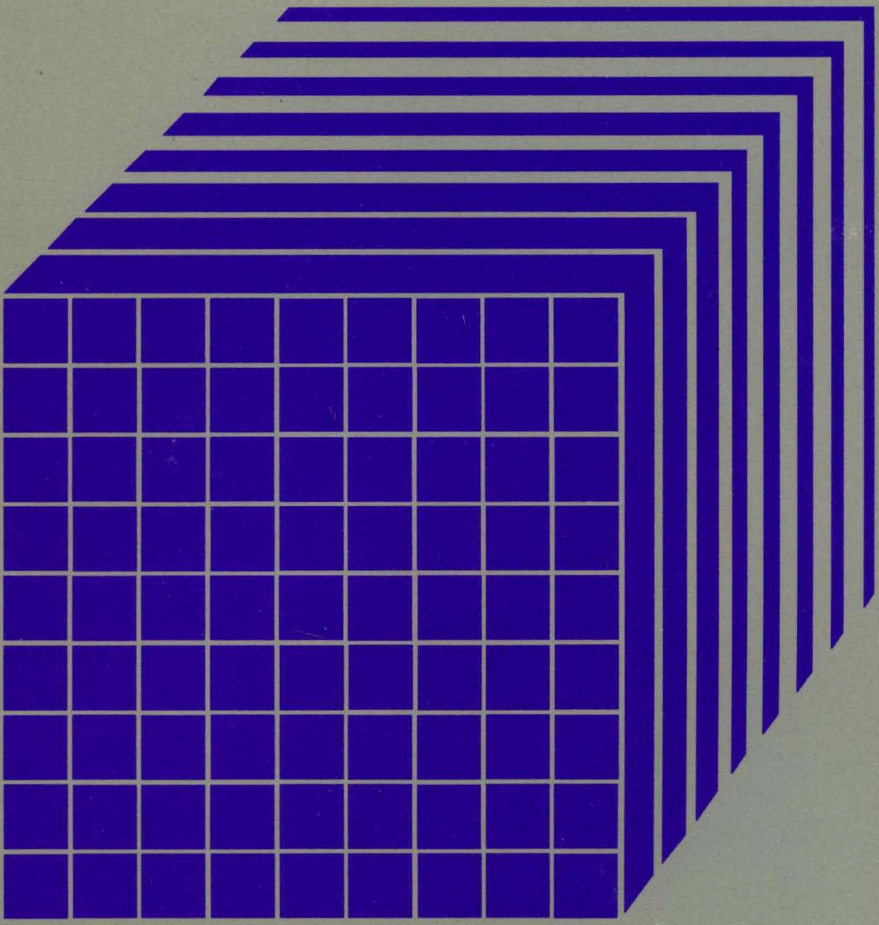
Virtual Machine/ System Product

System Logic and Problem Determination Guide Volume 2 (CMS)

Release 5

LY20-0893-4

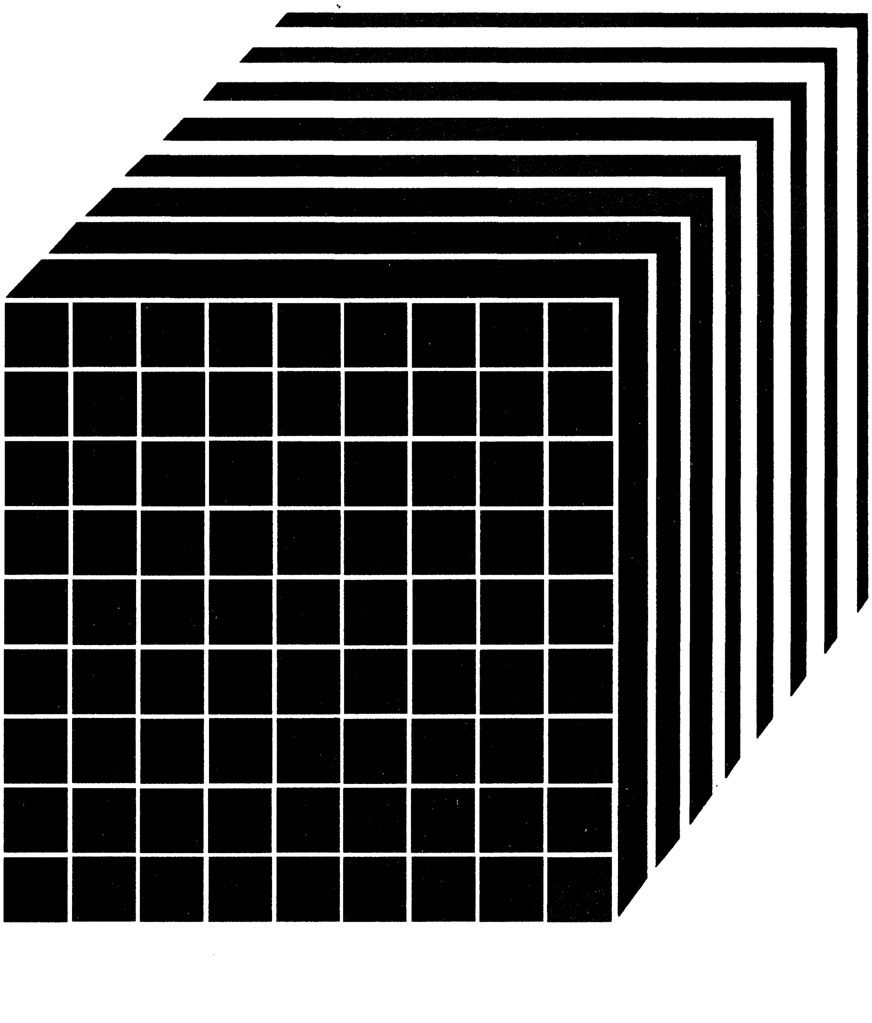


Virtual Machine/
System Product

System Logic and Problem Determination Guide Volume 2 (CMS)

Release 5

LY20-0893-4



Fifth Edition (December 1986)

This edition, LY20-0893-4, is a major revision of LY20-0893-3 and applies to Release 5 of the Virtual Machine/System Product (VM/SP), Program Number 5664-167, and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

Summary of Changes

For a list of changes, see page 299.

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

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

Ordering Publications

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

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

Preface

This manual provides the IBM system hardware and software support personnel with the information needed to analyze problems that may occur on the IBM Virtual Machine/System Product (VM/SP) when used in conjunction with VM/370 Release 6.

How This Manual is Organized

This manual is one of two volumes:

- Volume 1. VM/SP Control Program (CP)
- Volume 2. VM/SP Conversation Monitor System (CMS).

Each volume contains logic descriptions for the designated components of VM/SP. Each of these volumes is divided into four parts: Introduction, Method of Operation, Directory, and Diagnostic Aids.

The method of operation and program organization part contains the functions and relationships of the program routines in VM/SP. They indicate the program operation and organization in a general way to serve as a guide in understanding VM/SP. They are not meant to be a detailed analysis of VM/SP programming and cannot be used as such.

The directory contains a table of the CMS modules and their entry points.

The diagnostic aids part contains additional information useful for determining the cause of a problem.

Appendix A, located in Volume 2, contains a description of the CMS macro library.

Appendix B, also located in Volume 2, describes the CMS/DOS macro library.

Appendix C, also located in Volume 2, describes CMS/DOS support modules.

Information on the Remote Spooling Communication Subsystem Networking (RSCS), Version 2 is contained in the:

> *IBM VM/SP Remote Spooling Subsystem Networking, Version 2: Diagnosis Reference*, LY24-5228.

The control blocks supportive of the RSCS Logic are contained in:

> *VM/SP Data Areas and Control Block Logic Volume 2 (CMS)*, LY24-5221.

How to Use this Manual

- Isolate the component of VM/370 in which the problem occurred.
- Use the list of restrictions in *VM/SP System Messages and Codes* to be certain that the operation that was being performed was valid.
- Use the directories, *VM/SP Data Areas and Control Block Logic Volume 1 (CP)*, and *VM/SP Data Areas and Control Block Logic Volume 2 (CMS)* to help you to isolate the problem.
- Use the method of operation and program organization part, if necessary, to understand the operation that was being performed.

Contents

Figures

Part 1: Introduction to CMS

This part contains the following information:

- Conversational Monitor System (CMS)
- Interrupt Handling in CMS
- Functional Information
- OS Macro Simulation Under CMS
- VSE Support Under CMS.

Chapter 1. Conversational Monitor System (CMS)

The Conversational Monitor System (CMS), the major subsystem of VM/SP, provides a comprehensive set of conversational facilities to the user. Several copies of CMS may run under CP, thus providing several users with their own time sharing systems. CMS is designed specifically for the VM/SP virtual machine environment.

Each copy of CMS supports a single user. This means that the storage area contains only the data pertaining to that user. Likewise, each CMS user has his own machine configuration and his own files. Debugging is simpler because the files and storage area are protected from other users.

Programs can be debugged from the terminal. The terminal is used as a printer to examine limited amounts of data. After examining program data, the terminal user can enter commands on the terminal that alters the program. This is the most common method used to debug programs that run in CMS.

CMS, operating with the VM/SP Control Program, is a time sharing system suitable for problem solving, program development, and general work. It includes several programming language processors, file manipulation commands, utilities, and debugging aids. Additionally, CMS provides facilities to simplify the operation of other operating systems in a virtual machine environment when controlled from a remote terminal. For example, CMS creates and modifies job streams and analyzes virtual printer output.

Part of the CMS environment is related to the virtual machine environment created by CP. Each user is completely isolated from the activities of all other users, and each machine where CMS executes has virtual storage available to it and virtual storage managed for it by CP. The CP commands are recognized by CMS. For example, the commands allow messages to be sent to the operator or to other users and allow virtual devices to be dynamically detached from the virtual machine configuration.

The CMS Command Language

The CMS command language offers terminal users a wide range of functions. It supports a variety of programming languages, service functions, file manipulation, program execution control, and general system control. For detailed information on CMS commands, refer to the *VM/SP CMS Command Reference.*

Figure 10 on page 86 describes CMS command processing.

The File System

The Conversational Monitor System interfaces with virtual disks, tapes, and unit record equipment. The CMS residence device is a read-only, shared system disk. Permanent user files may be accessed from up to 25 active disks. CMS controls the logical access to these virtual disks, while CP facilities manage the device sharing and virtual-to-real mapping.

User files in CMS are identified with three designators. The first is filename. The second is filetype. The filetype may imply specific file characteristics to the CMS file management routines. The third is filemode. The filemode describes the location and access mode of the file.

The compilers available under CMS default to particular input filetypes, such as ASSEMBLE, but the file manipulation commands and listing commands do not default to a specific filetype. Files of a particular filetype form a logical data library for a user. For example, the collection of all COBOL source files, all object (TEXT) decks, or all EXEC procedures. This allows selective handling of specific groups of files with minimum input by the user.

User files can be created and changed directly from the terminal with the VM/SP System Product Editor. The VM/SP System Product Editor provides extensive context editing services. File characteristics such as record length, record format, tab locations, and serialization options can be specified. The VM/SP System Product Editor also provides fullscreen support for 3270 display stations.

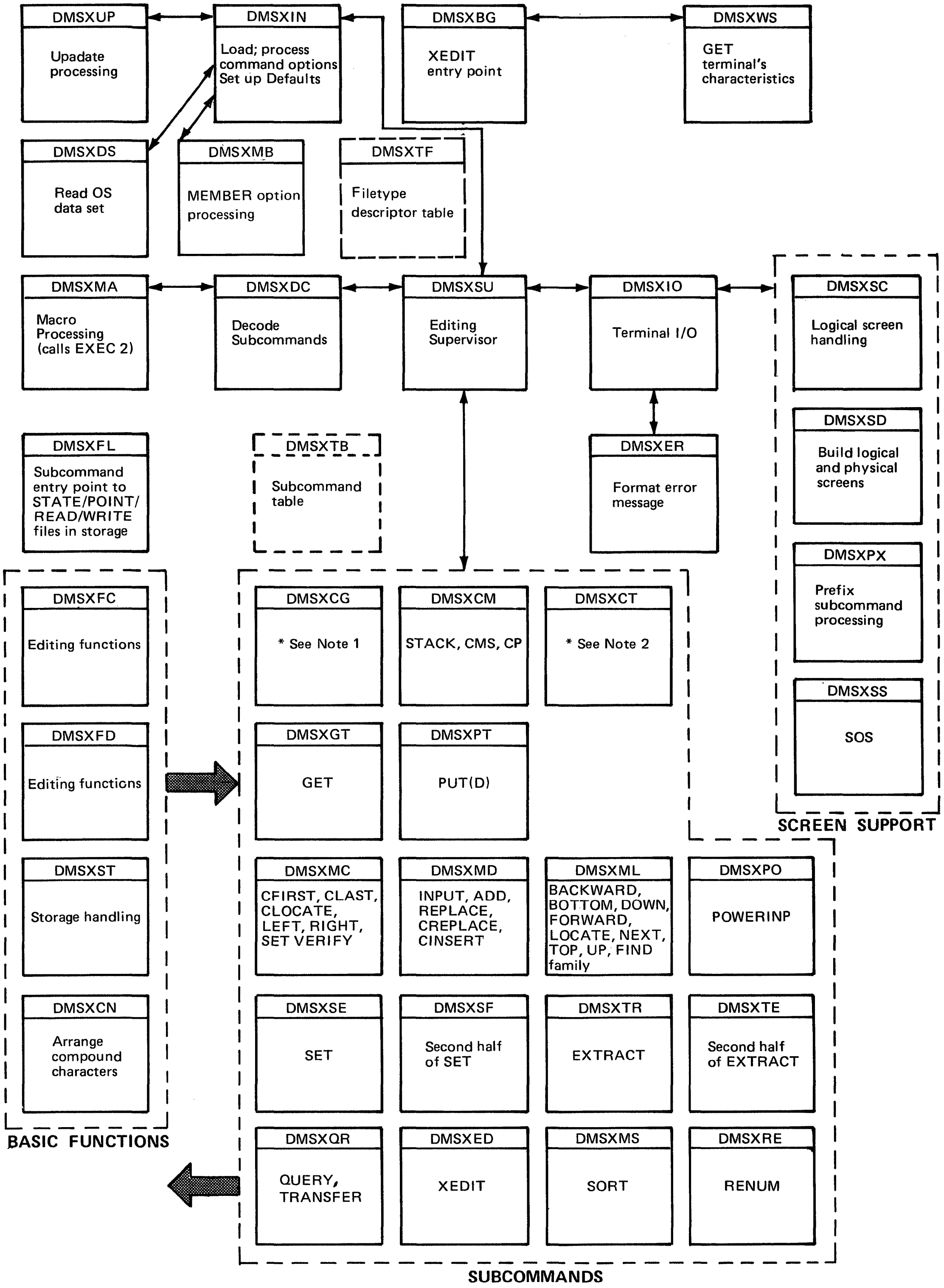
The major highlights of this editor include:

- Multiple views of the same or different files
- Selective column viewing
- Automatic wrapping of lines larger than the screen
- Ability to issue selected commands directly from the displayed line
- Ability to define screen format
- Extended string search functions
- Column pointing for editing within a line
- Ability to edit members contained in a CMS library
- Ability to manipulate files containing Double-Byte Character Set (DBCS) strings.

Additionally, the VM/SP System Product Editor provides language expansions and flexibility through the System Product Interpreter and the EXEC 2 processor. Figure 1 on page 6 describes the modules that perform the processing for the System Product Editor.

CMS automatically allocates compiler work files at the beginning of command execution on whichever active disk has the greatest amount of available space, and then CMS deallocates them at completion. Compiler object decks and listing files are normally allocated on the same disk as the input source file or on the primary read/write disk, and they are identified by combining the input filename with the filetypes TEXT and LISTING. These disk locations may be overridden by the user.

CMS disk files contain records stored on disks as 512-, 800-, 1024-, 2048-, or 4096-byte records. For disks with a blocksize of 800 bytes, a single user file is limited to a maximum of 65,533 records and must reside on one virtual disk. The file management system limits the number of files on the virtual disk to 3400. For disks with a blocksize of 512, 1024, 2048, and 4096 bytes, a single user file is limited to a maximum of 2^{31}-1 CMS blocks and must reside on one virtual disk. The maximum number of data blocks available in a variable format file on a 512-byte blocksize minidisk is about 15 times less than 2^{31}-1. This number is the maximum number of data blocks that can be accessed by the CMS file system due to the 5 level tree structure. The maximum number of files on any one disk is limited by the file management system to 2^{31}-1. However, the actual number of files is limited by the available disk space and the size of the user files.



***Note 1.** CDELETE, CHANGE, COMPRESS, COPY, COUNT, COVERLAY, DELETE, DUPLICATE, EXPAND, LOWERCAS, MERGE, MOVE, OVERLAY, RECOVER, SHIFT, UPPERCAS.

***Note 2.** CMSG, CURSOR, EMSG, FILE, LPREFIX, MSG, PFILE, PRESERVE, PSAVE, PURGE, READ, REFRESH, RENUM, REPEAT, RESET, RESTORE, SAVE, SET POINT, SET SCREEN, SET TERMINAL, TYPE.

Figure 1. Module Flow for the VM/SP System Product Editor

All CMS disk files are written as 512-, 800-, 1024-, 2048-, or 4096-byte records chained together by a specific master file entry that is stored in a table called the file directory. A separate file directory is kept for, and on, each virtual disk. The data records may be discontiguous, and they are allocated and deallocated automatically. A subset of the file directory (called the user file directory) is made resident in virtual storage when the disk directory is made available to CMS. It is updated on the virtual disk at least once per CMS command if the status of any file on that disk has been changed.

Virtual disks may be shared by CMS users. The capability is provided by VM/SP to all virtual machines, although a user interface is directly available in CMS commands. Specific files may be spooled between virtual machines to accomplish file transfer between users. Commands allow such file manipulations as writing from an entire disk or from a specific disk file to a tape, printer, punch, or the terminal. Other commands write from a tape or virtual card reader to disk, rename files, copy files, and erase files. Special macro libraries and text or program libraries are provided by CMS, and special commands are provided to update and use them. CMS files can be written onto and restored from unlabeled tapes via CMS commands.

Caution: Multiple write access under CMS can produce unpredictable results.

Problem programs that execute in CMS can create files on unlabeled tapes in any record and block size. The record format can be fixed, variable, or undefined. Figure 2 on page 8 describes the file system for an 800-byte record on disk. Figure 24 on page 135 shows the file system for 512-, 1K-, 2K-, and 4K-byte records on disk.

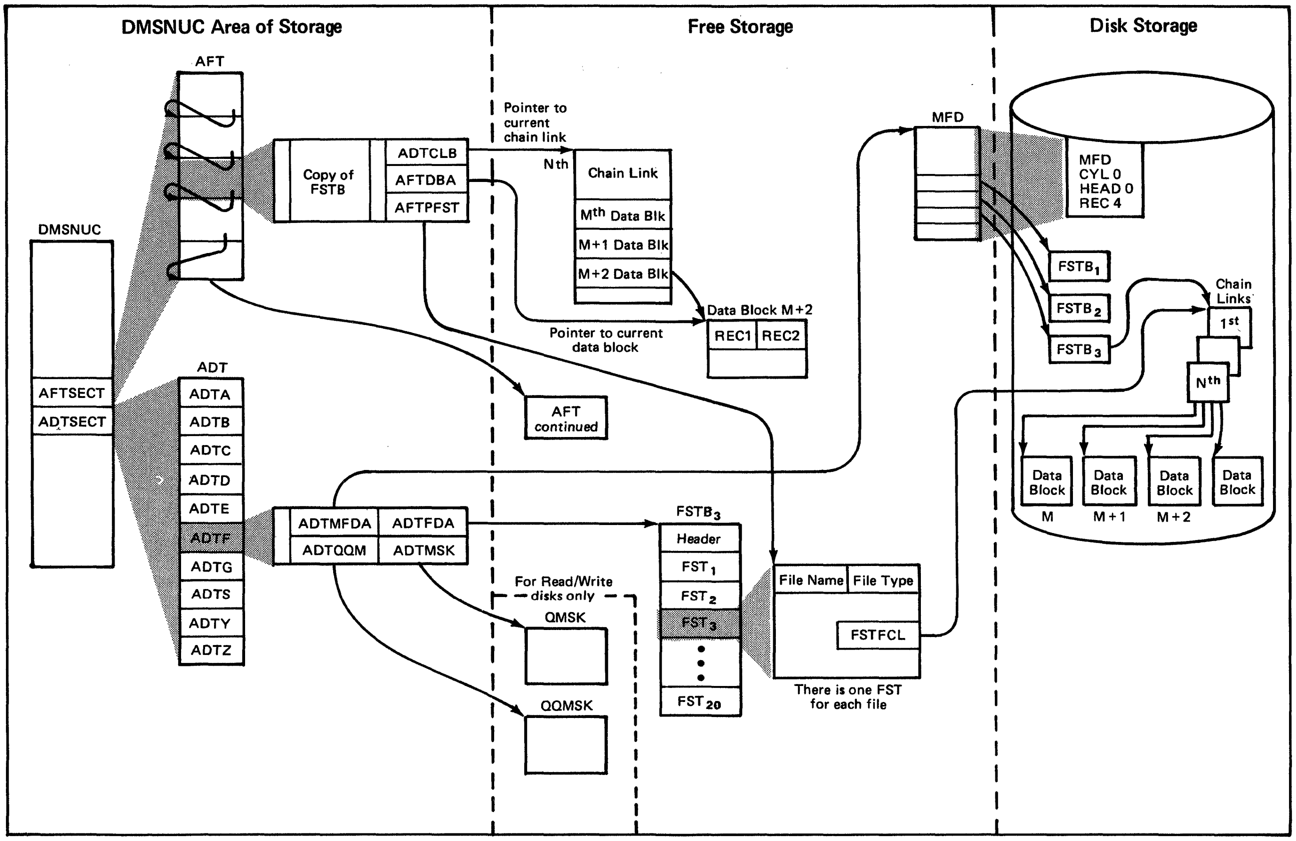


Figure 2. File System for an 800-Byte Record on Disk

Program Development

The Conversational Monitor System includes commands to create, compile, modify, and correct source programs; to build test files; to execute test programs; and to debug from the terminal. The commands of CMS are especially useful for OS and VSE program development, but the commands also may be used in combination with other operating systems to provide a virtual machine program development tool.

CMS uses the OS and VSE compilers via interface modules. The compilers themselves normally are not changed. To provide suitable interfaces, CMS includes a certain degree of OS and VSE simulation. For OS, the sequential, direct, and partitioned access methods are logically simulated. The data records are physically kept in the chained fixed-length blocks, and they are processed internally to simulate OS data set characteristics. For VSE, the sequential access method is supported. CMS supports VSAM catalogs, data spaces, and files on OS and DOS disks using the Access Method Services portion of VSE/VSAM. OS supervisor call functions such as GETMAIN/FREEMAIN and TIME are simulated.

For more information, about simulation restrictions in CMS see Chapter 4, "OS Simulation Under CMS" on page 27 and Chapter 5, "VSE Support Under CMS" on page 53.

Chapter 2. Interrupt Handling in CMS

CMS receives virtual SVC, input/output, machine, program, and external interruptions and passes control to the appropriate handling program.

SVC Interruptions

The Conversational Monitor System is SVC (supervisor call) driven. SVC interruptions are handled by the DMSITS resident routines. Two types of SVCs are processed by DMSITS: internal linkage SVC 202 and 203, and any other SVCs. The internal linkage SVC is issued by the command and function programs of the system when they require the services of other CMS programs. (Commands entered by the user from the terminal are converted to the internal linkage SVC by DMSINT). The OS SVCs are issued by the processing programs (for example, the Assembler).

Internal Linkage SVCs

When DMSITS receives control as a result of an internal linkage SVC (202 or 203), it saves the contents of the general registers, floating-point registers, and the SVC old PSW, establishes the normal and error return addresses, and passes control to the specified routine. (The routine is specified by the first 8 bytes of the parameter list whose address is passed in register 1 for SVC 202 or by a halfword code following SVC 203.)

For SVC 202, if the called program is not found in the internal function table of nucleus (resident) routines, DMSITS tries to call in a module (a CMS file with filetype MODULE) of this name via the LOADMOD command.

If the program was not found in the function table, nor was a module successfully loaded, DMSITS returns an error code to the caller.

To return from the called program, DMSITS restores the calling program's registers, and makes the appropriate normal or error return as defined by the calling program.

Other SVCs

The general approach taken by DMSITS to process other SVCs supported under CMS is essentially the same as that taken for the internal linkage SVCs. However, rather than passing control to a command or function program, as is the case with the internal linkage SVC, DMSITS passes control to the appropriate routine. The SVC number determines the appropriate routine.

In handling non-CMS SVC calls, DMSITS refers first to a user-defined SVC table (if one has been set up by the DMSHDS program). If the user-defined SVC table is present, any SVC number (other than 202 or 203) is looked for in that table. If it is found, control is transferred to the routine at the specified address.

If the SVC number is not found in the user-defined SVC table (or if the table is nonexistent), DMSITS either transfers control to the CMSDOS shared segment (if SET DOS ON has been issued) or DMSITS searches the standard system table (contained in DMSSVT) of OS calls for that SVC number. If the SVC number is found, control is transferred to the corresponding address in the usual manner. If the SVC is not in either table, the supervisor call is treated as an abend call.

The DMSHDS initialization programs sets up the user-defined SVC table. Then, the user can provide his own SVC routines.

Input/Output Interruptions

All input/output interruptions are received by the I/O interrupt handler, DMSITI. DMSITI saves the I/O old PSW and the CSW (channel status word). It then determines the status and requirements of the device causing the interruption and passes control to the routine that processes interruptions from that device.

DMSITI first scans CONSOLE function device entries (CDEV) until it finds one containing the device address that is the same as the interrupting device. If a matching device is found and a CONSOLE 'path' is waiting for an interrupt:

1. The wait field is cleared in the device entry,

2. The wait bit is turned off in the I/O old PSW, and

3. DMSITI returns control to the CONSOLE service by loading the I/O old PSW.

If no path is waiting, the interrupt is considered unsolicited and DMSITI checks for a user-defined interrupt handling routine. If DMSITI finds one, it passes control to the routine. Otherwise, if the device also exists in a console CDEV entry, DMSITI checks if any I/O was done and if an EXIT routine is specified. If an EXIT can be called, DMSITI turns off the PSW wait bit, loads the PSW, and exits.

If no console path performed I/O or no exits were called, the interrupt for the virtual console is passed to the system routine (DMSCITA) found in the CMS device table (DEVTAB). For dialed devices, the unsolicited interrupt is ignored. If fullscreen CMS is on, attention interrupts for the virtual console are passed to a fullscreen read routine instead of DMSCITA.

The device table (DEVTAB) contains an entry for each device in the system. Each entry for a particular device contains, among other things, the address of the program that processes interruptions from that device.

When the appropriate interrupt handling routine completes its processing, it returns control to DMSITI. At this point, DMSITI tests the wait bit in the saved I/O old PSW. If this bit is off, the interruption was probably caused by a terminal (asynchronous) I/O operation. DMSITI then returns control to the interrupted program by loading the I/O old PSW.

If the wait bit is on, the interruption was probably caused by a non-terminal (synchronous) I/O operation. The program that initiated the operation most likely called the DMSIOW function routine to wait for a particular type of interruption (usually a device end). In this case, DMSITI checks the pseudo-wait bit in the device table entry for the interruption device. If this bit is off, the system is waiting for some event other than the interruption from the interrupting device. DMSITI returns to the wait state by loading the saved I/O old PSW. (This PSW has the wait bit on.)

If the pseudo-wait bit is on, the system is waiting for an interruption from that particular device. If this interruption is not the one being waited for, DMSITI loads the saved I/O old PSW. This again places the machine in the wait state. Thus, the program that is waiting for a particular interruption is kept waiting until that interruption occurs.

If the interruption is the one being waited for, DMSITI resets both the pseudo-wait bit in the device table entry and the wait bit in the I/O old PSW. It then loads the PSW. This causes control to be returned to the DMSIOW function routine, which, in turn, returns control to the program that called it to wait for the interruption.

Terminal Interruptions

Terminal input/output interruptions are handled by the DMSCIT module. All interruptions other than those containing device end, channel end, attention, or unit exception status are ignored. If device end status is present with attention and a write CCW was terminated, its buffer is unstacked. An attention interrupt causes a read to be issued to the terminal, unless attention exits have been queued via the STAX macro. The attention exit with the highest priority is given control at each attention until the queue is exhausted; then a read is issued.

Device end status indicates that the last I/O operation has been completed. If the last I/O operation was a write, the line is deleted from the output buffer and the next write, if any, is started. If the last I/O operation was a normal read, the buffer is put on the finished read list and the next operation is started.

If the read is caused by an attention interrupt, the line is first checked to see if it is an immediate command (user-defined or built-in). If it is a user-defined immediate command, control is passed to a user specified exit, if one exists. Upon completion the exit returns to DMSCIT. If it is a built-in immediate command (HX, for example), appropriate processing is performed by DMSCIT.

Unit exception indicates a canceled read. The read is reissued, unless it had been issued with ATTREST=NO, in which case unit exception is treated as a device end.

Reader/Punch/Printer Interruptions

Interruptions from these devices are handled by the routines that actually issue the corresponding I/O operations. When an interruption from any of these devices occurs, control passes to DMSITI. The DMSITI passes control to DMSIOW, which returns control to the routine that issued the I/O operation. This routine can then analyze the cause of the interruption.

User-Controlled Device Interruptions

Interrupts from devices under user control are serviced the same as CMS devices except that DMSIOW and DMSITI manipulate a user-created device table, and DMSITI passes control to any user-written interrupt processing routine that is specified in the user device table. Otherwise, the processing program regains control directly.

- Users may now specify the exit parameter for the OPEN function of the CONSOLE macro instruction to handle unsolicited device interrupts. If this is specified, users should NOT define an interruption routine via the HNDINT macro for the same device. Use of the CONSOLE and HNDINT macros should be mutually exclusive. However, if for some reason there is both a CONSOLE exit and an HNDINT routine for the same device, the HNDINT routine overrides a CONSOLE exit only in the case of an unsolicited interrupt.

  CONSOLE supports multiple applications for a single device whereas HNDINT only allows one application to handle all interrupts from a specific device. Because it is difficult to tell what application is doing I/O last, CONSOLE helps CMS keep track of what application is doing I/O or what application handled interrupts last.

- CONSOLE OPEN with EXIT supersedes an HNDINT routine when the interrupt is solicited. Therefore, if users want to do I/O to a 3270 device, they should use the CONSOLE macro instead of the HNDINT macro.

Program Interruptions

The program interruption handler, DMSITP, receives control when a program interruption occurs. When DMSITP gets control, it stores the program old PSW and the contents of registers 14, 15, 0, 1, and 2 into the program interruption element (PIE). (The routine that handles the SPIE macro instruction has already placed the address of the program interruption control area (PICA) into PIE.) DMSITP then determines whether or not the event that caused the interruption was one of those selected by a SPIE macro instruction. If it was not, DMSITP passes control to the DMSABN abend recovery routine.

If the cause of the interruption was one of those selected in a SPIE macro instruction, DMSITP picks up the exit routine address from the PICA and passes control to the exit routine. Upon return from the exit routine, DMSITP returns to the interrupted program by loading the original program check old PSW. The address field of the PSW was modified by a SPIE exit routine in the PIE.

External Interruptions

An external interruption causes control to be passed to the external interrupt handler DMSITE. If CMS IUCV support is active in the virtual machine and an IUCV external interrupt occurs, control is passed to the user exit specified on the HNDIUCV or CMSIUCV macro. If the user has issued the HNDEXT macro to trap external interrupts, DMSITE passes control to the user's exit routine.

If the interrupt was caused by the timer, DMSITE resets the timer and types the BLIP character at the terminal. The standard BLIP timer setting is two seconds, and the standard BLIP character is uppercase, followed by the lowercase (it moves the typeball without printing). Otherwise, control is passed to the DEBUG routine.

Machine Check Interruptions

Hard machine check interruptions on the real processor are not reflected to a CMS virtual user by CP. A message prints on the console indicating the failure. The user is then disabled and must IPL CMS again in order to continue.

Chapter 3. Functional Information

The most important thing to remember about CMS, from a debugging standpoint, is that it is a one-user system. The supervisor manages only one user and keeps track of only one user's file and storage chains. Thus, everything in a dump of a particular machine relates only to that virtual machine's activity.

You should be familiar with register usage, save area structuring, and control block relationships before attempting to debug or alter CMS.

Register Usage

When a CMS routine is called, R1 must point to a valid parameter list (PLIST) for that program. On return, R0 may or may not contain meaningful information. For example, on return from a call to FILEDEF with no change, R0 contains a negative address if a new FCB (file control block) has been set up; otherwise, R0 contains a positive address of the already existing FCB. R15 contains the return code, if any. The use of registers 0 and 2 through 11 varies.

On entry to a command or routine called by SVC 202 the following are in effect:

Register	Contents
0	The address of EPLIST, if available
1	The address of the PLIST supplied by the caller
12	The address entry point of the called routine
13	The address of a work area (12 doublewords) supplied by SVCINT - the SVC handler
14	The return address to the SVC handler
15	The entry point (same as register 12)

On return from a routine, register 15 contains:

Return Code	Meaning
0	No error occurred
<0	Called routine not found
>0	Error occurred

If a CMS routine is called by an SVC 202, CMS saves and restores registers 0 through 14.

Most CMS routines use register 12 as a base register.

Structure of CMS Storage

Figure 3 on page 20, Figure 4 on page 21, and Figure 5 on page 22 describes how CMS uses its virtual storage. The pointers indicated (MAINSTRT, MAINHIGH, and FREELOWE) are all found in NUCON (the nucleus constant area).

The sections of CMS storage have the the following uses:

DMSNUC (X'00000' to ANUCEND).
This is the nucleus constant area. It contains pointers, flags, and other data updated by the various system routines.

Low-Storage DMSFREE User Free Storage Area (ANUCEND to X'0E000').
This area is a free storage area where user requests to DMSFREE are allocated.

Transient Program Area (X'0E000' to X'10000').
Since it is not essential to keep all nucleus functions resident in storage all the time, some of them are made "transient." This means that when nucleus functions are needed, they are loaded from the disk into the transient program area. Such programs may not be longer than two pages because that is the size of the transient area. (A page is 4096 bytes of virtual storage.) All transient routines must be serially reusable since they are not read in each time they are needed.

Low-Storage DMSFREE Nucleus Free Storage Area (X'10000' to X'20000').
This area is a free storage area where nucleus requests to DMSFREE are allocated. The top part of this area contains the dummy hyperblocks for the S and Y disks. Each block is 48 bytes long. This area may be followed by the file status tables for the S2 filemode files of the system disk.

If there is enough room, the FREETAB table also occupies this area, just below the file status tables, if they are there. Each entry in the FREETAB table is one byte long. Each byte represents one page (4K or 4096 bytes) of defined storage.

User Program Area (X'20000' to Loader Tables or CMS Nucleus, whichever has the lowest value).
User programs are loaded into this area by the LOAD command for text decks or by the LOADMOD command for modules. Storage allocated by means of the GETMAIN macro instruction is taken from this area, starting from the high address of the user program. In addition, this storage area can be allocated from the top down by DMSFREE, if there

is not enough storage available in the low DMSFREE storage area. Thus, the usable size of the user program area is reduced by the amount of free storage that has been allocated from it by DMSFREE.

Loader Tables (Top pages of storage).

The top of storage is occupied by the loader tables, which are required by the CMS loader. These tables indicate which modules are currently loaded in the user program area (and the transient program area after a LOAD command). The size of the loader tables can be varied by the SET LDRTBLS command. However, to successfully change the size of the loader tables, the SET LDRTBLS command should be issued immediately after IPL. If SET LDRTBLS is not issued immediately, high storage may be fragmented.

CMS Nucleus.

The CMS nucleus contains the reentrant code for the CMS nucleus routines and the system S-STAT and Y-STAT. If there is not sufficient room to contain the S-STAT in this area, it is placed in low DMSFREE nucleus storage. If there is not sufficient room to contain the Y-STAT in this area, the Y-disk is accessed using the ACCESS command.

If the size of the user's virtual machine is defined below the end of the CMS nucleus (refer to label NUCSIGMA in Figure 4 on page 21), it is not possible to IPL by device name. You cannot IPL by device name because the CMS nucleus is too large to be loaded into the user's virtual storage. Therefore, the user can only IPL by the system name (such as, IPL CMS). The loader table is placed immediately below the CMS nucleus.

On the other hand, if the size of the user's virtual machine is defined above the end of the CMS nucleus (see Figure 4 on page 21 and Figure 5 on page 22), the user may IPL by either device name or system name.

IPLing by device name:

The S-STAT, Y-STAT, and the loader table are placed above the CMS nucleus. If there is not enough room to contain the S-STAT above the CMS nucleus (NUCSIGMA), it is placed in low storage. Likewise, if there is not sufficient room for the loader table above the CMS nucleus (NUCSIGMA), the loader table is placed below the nucleus. Any leftover free space above the nucleus is placed on the high DMSFREE chain.

IPLing by system name:

The shared copy of the S-STAT, Y-STAT and nucleus is used. If there is sufficient room, the loader table is placed above the S-STAT and Y-STAT (NUCOMEGA). If there is not sufficient room to place the loader table above the S-STAT and Y-STAT, the loader table is placed below the nucleus. Any leftover free space above the S-STAT and Y-STAT (NUCOMEGA) is placed on the high DMSFREE chain.

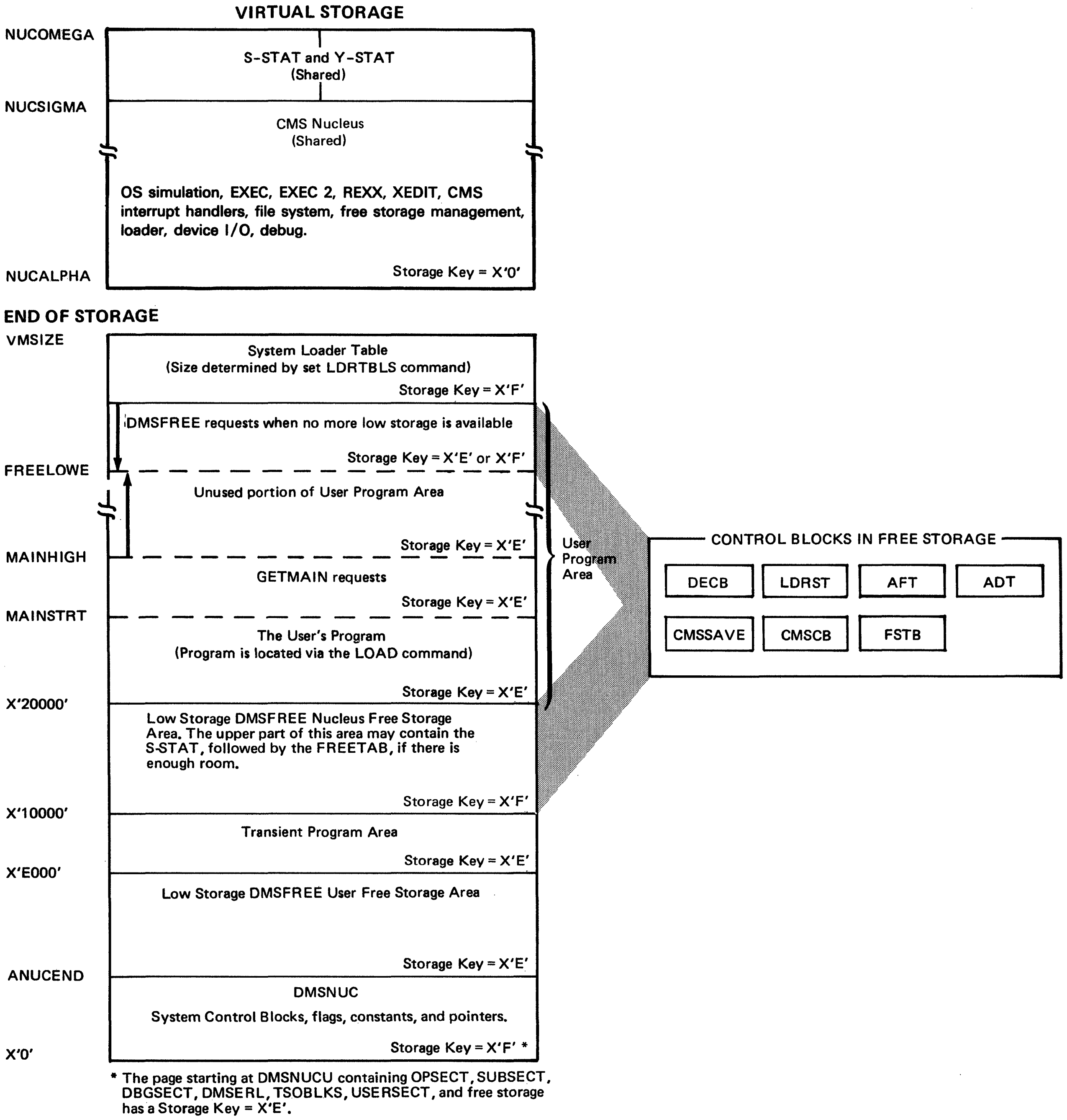
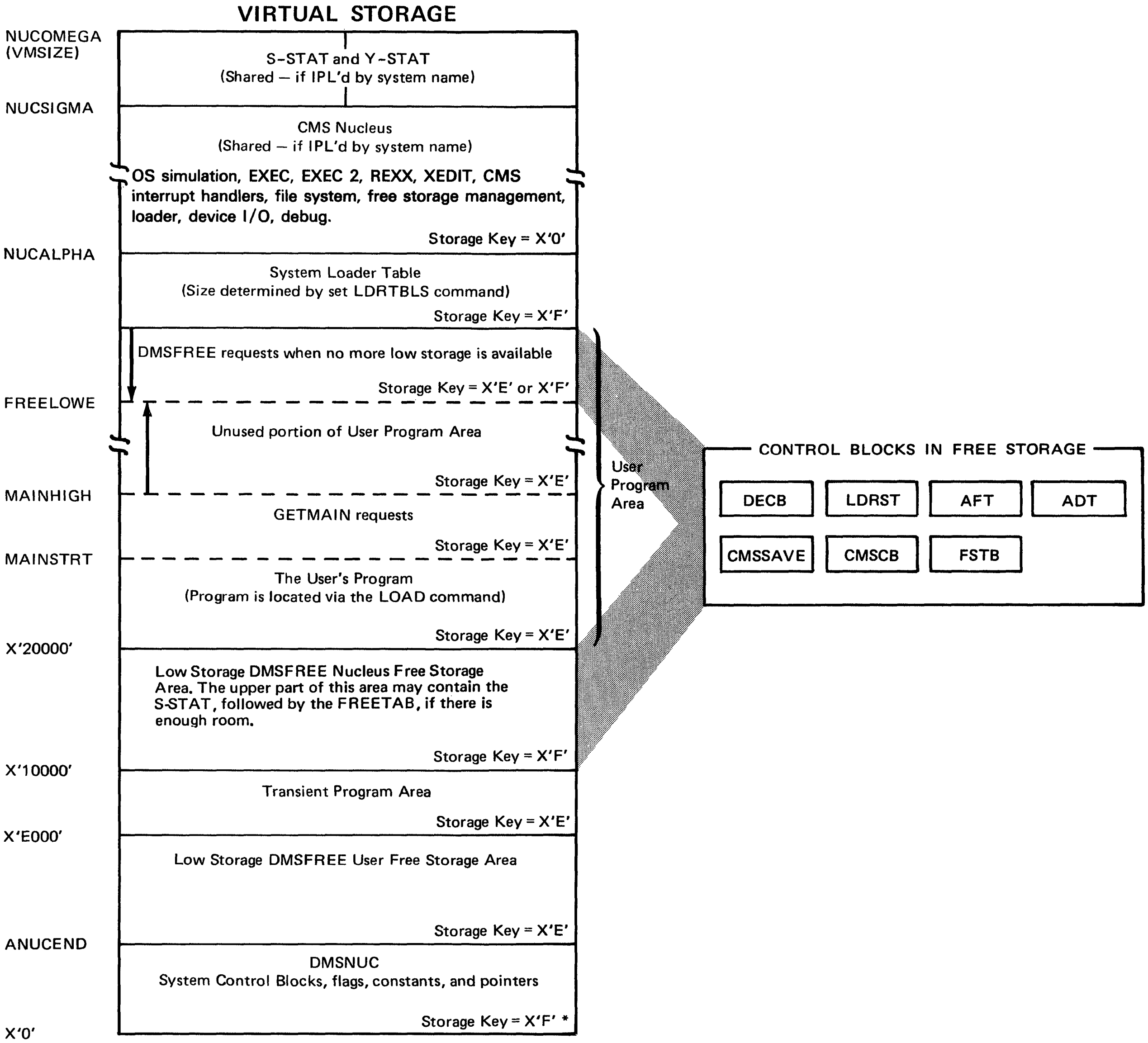
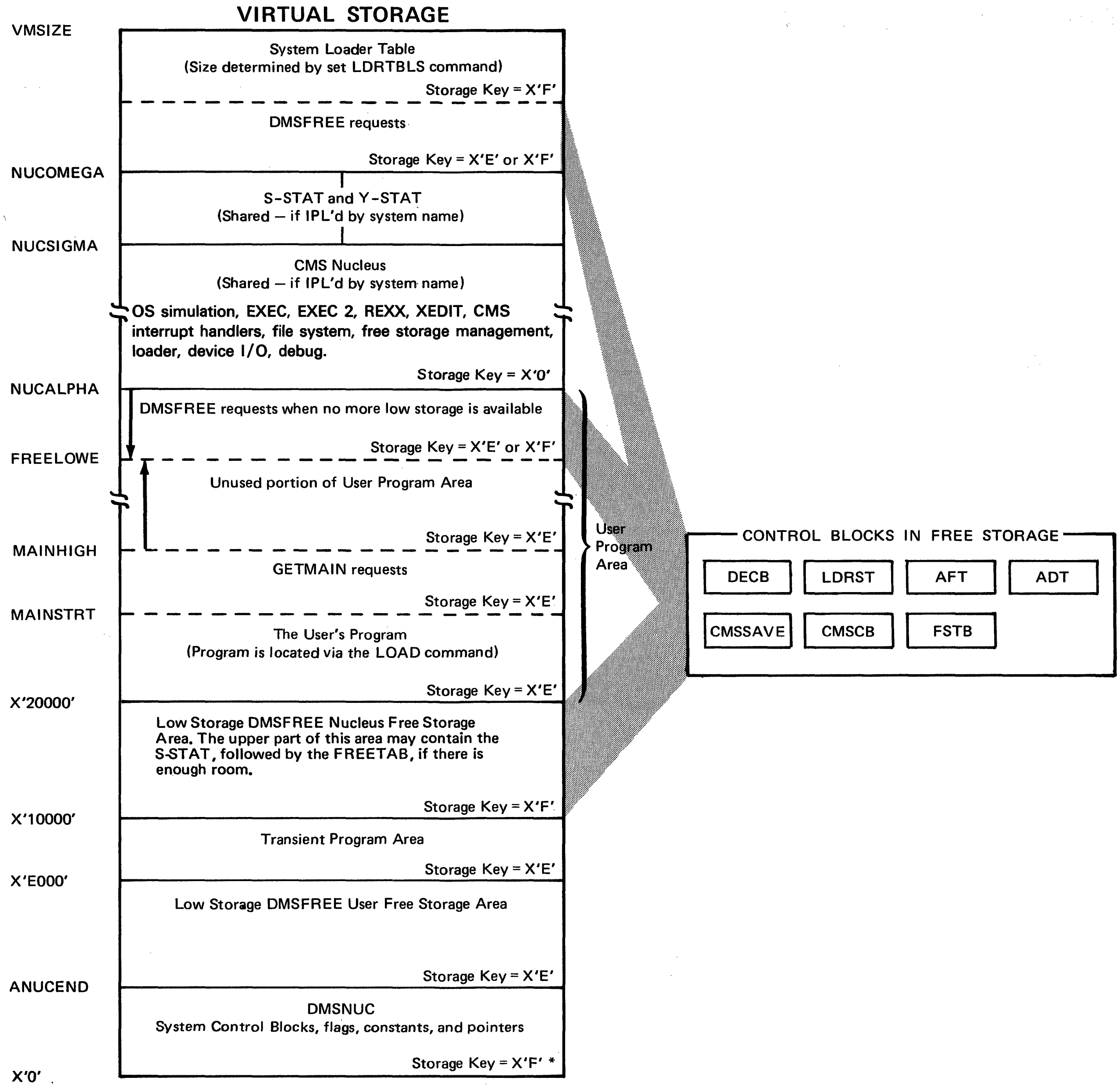

VIRTUAL STORAGE
NUCOMEGA
S-STAT and Y-STAT
(Shared)
NUCSIGMA
CMS Nucleus
(Shared)
OS simulation, EXEC, EXEC 2, REXX, XEDIT, CMS interrupt handlers, file system, free storage management, loader, device I/O, debug.
NUCALPHA
Storage Key = X'0'
END OF STORAGE
VMSIZE
System Loader Table
(Size determined by set LDRTBLS command)
Storage Key = X'F'
DMSFREE requests when no more low storage is available
Storage Key = X'E' or X'F'
FREELOWE
Unused portion of User Program Area
Storage Key = X'E'
MAINHIGH
GETMAIN requests
Storage Key = X'E'
MAINSTRT
The User's Program
(Program is located via the LOAD command)
Storage Key = X'E'
User Program Area
X'20000'
Low Storage DMSFREE Nucleus Free Storage Area. The upper part of this area may contain the S-STAT, followed by the FREETAB, if there is enough room.
Storage Key = X'F'
X'10000'
Transient Program Area
Storage Key = X'E'
X'E000'
Low Storage DMSFREE User Free Storage Area
Storage Key = X'E'
ANUCEND
DMSNUC
System Control Blocks, flags, constants, and pointers.
Storage Key = X'F' *
X'0'
CONTROL BLOCKS IN FREE STORAGE
DECB
LDRST
AFT
ADT
CMSSAVE
CMSCB
FSTB
* The page starting at DMSNUCU containing OPSECT, SUBSECT, DBGSECT, DMSERL, TSOBLKS, USERSECT, and free storage has a Storage Key = X'E'.

Figure 3. **CMS Storage Map 1.** CMS virtual storage usage when the CMS nucleus is larger than the user's virtual storage. In this case, you must IPL by system name (VMSIZE is less than NUCSIGMA). The arrows indicate that MAINHIGH is extended upward and FREELOWE is extended downward.


VIRTUAL STORAGE
NUCOMEGA (VMSIZE)
S-STAT and Y-STAT
(Shared – if IPL'd by system name)
NUCSIGMA
CMS Nucleus
(Shared – if IPL'd by system name)
OS simulation, EXEC, EXEC 2, REXX, XEDIT, CMS interrupt handlers, file system, free storage management, loader, device I/O, debug.
Storage Key = X'0'
NUCALPHA
System Loader Table
(Size determined by set LDRTBLS command)
Storage Key = X'F'
DMSFREE requests when no more low storage is available
Storage Key = X'E' or X'F'
FREELOWE
Unused portion of User Program Area
Storage Key = X'E'
MAINHIGH
GETMAIN requests
Storage Key = X'E'
MAINSTRT
The User's Program
(Program is located via the LOAD command)
Storage Key = X'E'
User Program Area
X'20000'
Low Storage DMSFREE Nucleus Free Storage Area. The upper part of this area may contain the S-STAT, followed by the FREETAB, if there is enough room.
Storage Key = X'F'
X'10000'
Transient Program Area
Storage Key = X'E'
X'E000'
Low Storage DMSFREE User Free Storage Area
Storage Key = X'E'
ANUCEND
DMSNUC
System Control Blocks, flags, constants, and pointers
Storage Key = X'F' *
X'0'
CONTROL BLOCKS IN FREE STORAGE
DECB
LDRST
AFT
ADT
CMSSAVE
CMSCB
FSTB

* The page starting at DMSNUCU containing OPSECT, SUBSECT, DBGSECT, DMSERL, TSOBLKS, USERSECT, and free storage has a Storage Key = X'E'.

Figure 4. **CMS Storage Map 2.** Virtual storage usage when the user's virtual storage is equal to the CMS nucleus. The user may IPL by system name or device. In addition, this figure shows where there is insufficient room to place the loader table above S-STAT and Y-STAT. The arrows indicate that MAINHIGH is extended upward and FREELOWE is extended downward.


VIRTUAL STORAGE
VMSIZE
System Loader Table
(Size determined by set LDRTBLS command)
Storage Key = X'F'
DMSFREE requests
Storage Key = X'E' or X'F'
NUCOMEGA
S-STAT and Y-STAT
(Shared – if IPL'd by system name)
NUCSIGMA
CMS Nucleus
(Shared – if IPL'd by system name)
OS simulation, EXEC, EXEC 2, REXX, XEDIT, CMS interrupt handlers, file system, free storage management, loader, device I/O, debug.
Storage Key = X'0'
NUCALPHA
DMSFREE requests when no more low storage is available
Storage Key = X'E' or X'F'
FREELOWE
Unused portion of User Program Area
Storage Key = X'E'
MAINHIGH
GETMAIN requests
Storage Key = X'E'
MAINSTRT
The User's Program
(Program is located via the LOAD command)
Storage Key = X'E'
User Program Area
X'20000'
Low Storage DMSFREE Nucleus Free Storage Area. The upper part of this area may contain the S-STAT, followed by the FREETAB, if there is enough room.
Storage Key = X'F'
X'10000'
Transient Program Area
Storage Key = X'E'
X'E000'
Low Storage DMSFREE User Free Storage Area
Storage Key = X'E'
ANUCEND
DMSNUC
System Control Blocks, flags, constants, and pointers
Storage Key = X'F' *
X'0'
CONTROL BLOCKS IN FREE STORAGE
DECB
LDRST
AFT
ADT
CMSSAVE
CMSCB
FSTB

* The page starting at DMSNUCU containing OPSECT, SUBSECT, DBGSECT, DMSERL, TSOBLKS, USERSECT, and free storage has a Storage Key = X'E'.

Figure 5. **CMS Storage Map 3.** CMS virtual storage usage when the user's virtual storage is larger than the CMS nucleus. The user may IPL by system name or device. In addition, this figure shows where there is sufficient room to place the system loader table above S-STAT and Y-STAT. The arrows indicate that MAINHIGH is extended upward and FREELOWE is extended downward.

Structure of DMSNUC

DMSNUC is the portion of storage in a CMS virtual machine that contains system control blocks, flags, constants, and pointers.

The CSECTs in DMSNUC contain only symbolic references. This means that an update or modification to CMS, which changes a CSECT in DMSNUC, does not automatically force all CMS modules to be recompiled. Only those modules that refer to the area that was redefined must be recompiled.

USERSECT (User Area)

The USERSECT CSECT defines space that is not used by CMS. A modification or update to CMS can use the 18 fullwords defined for USERSECT. There is a pointer (AUSER) in the NUCON area to the user space.

DEVTAB (Device Table)

The DEVTAB CSECT is a table describing the devices available for the CMS system. The table contains the following entries:

- 1 console
- 26 disks
- 1 reader
- 1 punch
- 1 printer
- 16 tapes
- 1 dummy.

You can change some existing entries in DEVTAB. Each device table entry contains the following information:

- Virtual device address
- Device flags
- Device types
- Symbol device name
- Address of the interrupt processing routine (for the console).

The virtual address of the console is defined at logon time. The ACCESS command can dynamically alter the virtual address of the user disks in DEVTAB. The virtual address of a tape can be reassigned to any of the addresses given in DEVTAB (TAP0 - TAPF) by using CMS commands and/or macros. Changing the virtual addresses of the reader, printer, or punch in DEVTAB has no effect.

CMS Interface for Display Terminals

CMS has an interface allowing it to display large amounts of data in a very rapid fashion. This interface for 3270 display terminals (also 3138, 3148, and 3158) is much faster and has less overhead than the normal write because it displays up to 1760 characters in one operation, instead of issuing 22 individual writes of 80 characters each (that is one write per line on a display terminal). Data displayed in the screen output area with this interface is not placed in the console spool file.

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

The CONSOLE macro allows programs to open 'paths' to a display device. A path is a unique name that distinguishes one application from another and allows the console facility to coordinate the use of the screen. For example, if an application is writing to the screen, the CONSOLE macro tells it that another 'path' has updated the screen lastly, and, therefore, the screen must be reformatted. Because of this, fullscreen applications do not have to rewrite the entire screen every time a write is done.

Screen coordination can be done only for applications using the console facility. Because some application still issue their own DIAGNOSE code X'58', you must reformat the screen. This avoids mixing data from two different applications on the screen.

The CONSOLE macro provides the following functions:

- OPEN/CLOSE - Opening and closing a specific path to the console.
- READ/WRITE - Reading and writing buffers that have 3270 data streams built by the application. In order to write to the screen, applications must construct a valid 3270 data stream. When a read is performed, the data is returned in the user's buffer. The CMS console facility issues the DIAGNOSE code X'58' for the virtual console or a Start I/O (SIO) for dialed devices, builds the CCW for READ and WRITE requests, tests conditions after I/O, and gives the result of the I/O operation to the application.
- EXCP - Performing READ or WRITE I/O operations using CCWs that applications supply. An application ***must*** supply its own CCW if it uses the EXCP function. This function is intended for use with dialed devices.
- WAIT - Wait for an I/O interrupt from the console device.
- QUERY - Getting information about the device attributes (DIAGNOSE code X'24' and DIAGNOSE code X'8C'), or if the path is opened,

getting information about a specific path and its associated device. The user should provide a buffer for this information and then map the information using the CQYSECT mapping macro. For information about the CQYSECT macro, refer to *VM/SP Data Areas and Control Block Logic Volume 2 (CMS).*

The four formats of the CONSOLE macro instruction are:

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

Note: For the detailed formats of the CONSOLE macro, see *VM/SP CMS Macros and Functions Reference.*

Although the CONSOLE macro is the preferred interface for fullscreen I/O, the DISPW macro may be used to generate a calling sequence for the CMS display terminal interface module, DMSGIO. DMSGIO creates a channel program and issues a DIAGNOSE code X'58' to display the data. DMSGIO is a TEXT file that must be loaded to use DISPW.

The format of the CMS DISPW macro is:

[*label*]	**DISPW**	*bufad* $\left[\textbf{,LINE=}\left\{\begin{array}{l} n \\ \underline{\mathbf{0}} \end{array}\right\}\right]\left[\textbf{,BYTES=}\left\{\begin{array}{l} nnnn \\ \underline{\mathbf{1760}} \end{array}\right\}\right]$ [**,ERASE=YES**] [**,CANCEL=YES**]

where:

label
is an optional macro statement label.

bufad
is the address of a buffer containing the data to be written to the display terminal.

LINE= $\left\{\begin{array}{l} n \\ \underline{0} \end{array}\right\}$
is the number of the line, 0 to 23, on the display terminal that is to be written. Line number 0 is the default.

BYTES= $\left\{\begin{array}{l} \boldsymbol{nnnn} \\ \underline{\mathbf{1760}} \end{array}\right\}$
is the number of bytes (0 to 1760) to be written on the display terminal. 1760 bytes is the default.

ERASE = YES

specifies that the display screen is to be erased before the current date is written. The screen is erased regardless of the line or number of bytes to be displayed. Specifying ERASE = YES causes the screen to go into "MORE" status.

CANCEL = YES

causes the CANCEL operation to be performed: the output area is erased.

Note: It is advisable for the user to save registers before issuing the DISPW macro and to restore them after the macro because the modules called by the DISPW macro do not save the user's registers. The DISPW macro saves and restores register 13.

Chapter 4. OS Simulation Under CMS

When a language processor or a user-written program is executing in the CMS environment and using OS-type functions, it is not executing OS code. Instead, CMS provides routines that simulate the OS functions required to support OS language processors and their generated object code.

CMS functionally simulates the OS macros in a way that presents equivalent results to programs executing under CMS. The OS macros are supported only to the extent stated in the publications for the supported language processors, and then only to the extent necessary to successfully satisfy the specific requirement of the supervisory function.

OS Data Management Simulation

The disk format and data base organization of CMS are different from those of OS. A CMS file produced by an OS program running under CMS and written on a CMS disk has a different format from that of an OS data set produced by the same OS program running under OS and written on an OS disk. The data is exactly the same, but its format is different. (An OS disk is formatted by an OS program, such as Device Support Facility.) CMS does not support multi-buffering for unit record devices. There is one DCB per device, not per file.

Handling Files that Reside on CMS Disks

CMS can read, write, or update any OS data that resides on a CMS disk. By simulating OS macros, CMS simulates the following access methods so that OS data organized by these access methods can reside on CMS disks:

- BDAM (direct) -- identifying a record by a key or by its relative position within the data set.
- BPAM (partitioned) -- seeking a named member within data set.

  Note: Two BPAM files with the same filetype cannot be updated at the same time.
- BSAM/QSAM (sequential) -- accessing a record in a sequence in relation to preceding or following records.

- VSAM (direct or sequential) -- accessing a record sequentially or directly by key or address.

  Note: CMS support of OS VSAM files is based on VSE/VSAM. Therefore, the OS user is restricted to those functions available under VSE/VSAM.

Refer to Figure 6 and "OS Macros" on page 30, then read "Access Method Support" on page 38 to see how CMS handles these access methods.

Since CMS does not simulate the indexed sequential access method (ISAM), no OS program that uses ISAM can execute under CMS. Therefore, no program can write an indexed sequential data set on a CMS disk.

Handling Files that Reside on OS or DOS Disks

By simulating OS macros, CMS can read, but not write or update, OS sequential and partitioned data sets that reside on OS disks. Using the same simulated OS macros, CMS can read DOS sequential files that reside on DOS disks. The OS macros handle the DOS data as if it were OS data. Thus, a DOS sequential file can be used as input to an OS program running under CMS.

However, an OS sequential or partitioned data set that resides on an OS disk can be written or updated only by an OS program running in a real OS machine.

CMS can execute programs that read and write VSAM files from OS programs written in the VS BASIC, COBOL, PL/I, VS/APL, and VS FORTRAN programming languages. CMS also supports VSAM for use with DOS/VS SORT/MERGE. This CMS support is based on the VSE/VSAM program product, and, therefore, the OS user is limited to those VSAM functions that are available under VSE/VSAM.

OS Macro Simulation

The following figure shows the OS macro functions that are partially or completely simulated, as defined by SVC number:

Macro Name	SVC Number	Function
XDAP	00	Reads or writes direct access volumes
EXCP	00	Executes graphic channel programs for graphic access method (GAM)
WAIT	01	Waits for an I/O completion

Figure 6 (Part 1 of 3). Simulated OS Supervisor Calls

Macro Name	SVC Number	Function
POST	02	Posts the I/O completion
EXIT/RETURN	03	Returns from a called phase
GETMAIN	04	Conditionally acquires user storage
FREEMAIN	05	Releases user-acquired storage
GETPOOL	-	Simulates as SVC 10
FREEPOOL	-	Simulates as SVC 10
LINK	06	Links control to another phase
XCTL	07	Deletes, then links control to another load phase
LOAD	08	Reads a phase into storage
DELETE	09	Deletes a loaded phase
GETMAIN/ FREEMAIN	10	Manipulates user free storage
TIME	11	Gets the time of day
ABEND	13	Terminates processing
SPIE	14	Allows processing program to handle program interrupts
RESTORE	17	Effective NOP
BLDL	18	Builds a directory for a partitioned data set
FIND	18	Locates a member of a partitioned data set
OPEN	19	Activates a data file
CLOSE	20	Deactivates a data file
STOW	21	Manipulates partitioned directories
OPENJ	22	Activates a data file
TCLOSE	23	Temporarily deactivates a data file
DEVTYPE	24	Gets device-type physical characteristics
TRKBAL	25	Effective NOP
FEOV	31	Sets forced EOV error code
WTO/WTOR	35	Communicates with the terminal
EXTRACT	40	Effective NOP
IDENTIFY	41	Adds entry to loader table
ATTACH	42	Effective LINK
CHAP	44	Effective NOP
TTIMER	46	Accesses or cancels timer
STIMER	47	Sets timer interval and timer exit routine
DEQ	48	Effective NOP
SNAP	51	Dumps specified areas of storage
ENQ	56	Effective NOP
FREEDBUF	57	Releases a free storage buffer

Figure 6 (Part 2 of 3). Simulated OS Supervisor Calls

Macro Name	SVC Number	Function
STAE	60	Allows processing program to decipher abend conditions
DETACH	62	Effective NOP
CHKPT	63	Effective NOP
RDJFCB	64	Obtains information from FILEDEF command
SYNAD	-	Handles data set error conditions
SYNADAF	68	Provides SYNAD analysis function
SYNADRLS	68	Releases SYNADAF message and save areas
BSP	69	Backs up a record on a tape or disk
TGET/TPUT	93	Reads or writes a terminal line
TCLEARQ	94	Clears terminal input queue
STAX	96	Updates a queue of CMTAXEs that creates an attention exit block
PGRLSE	112	Releases storage contents
CALL	-	Transfers control to a control section at a specified entry
SAVE	-	Saves program registers
RETURN	-	Returns from a subroutine
GET/PUT	-	Reads/Writes system-blocked data (QSAM)
READ	-	Accesses system-record data
WRITE	-	Write system-record data
NOTE/POINT	-	Manages data set positioning
CHECK	-	Verifies READ/WRITE completion
DCB	-	Constructs a data control block
DCBD	-	Generates a DSECT for a data control block

Figure 6 (Part 3 of 3). Simulated OS Supervisor Calls

OS Macros

Because CMS has its own file system and is a single-user system operating in a virtual machine with virtual storage, there are certain restrictions for the simulated OS function in CMS. For example, HIARCHY options and options that are used only by OS multi-tasking systems are ignored by CMS.

Due to the design of the CMS loader, an XCTL from the explicitly loaded phase, followed by a LINK by succeeding phases, may cause unpredictable results.

Listed below are descriptions of all the OS macro functions that are simulated by CMS as seen by the programmer. Implementation and program results that differ from those given in *OS Data Management Macro Instructions* and *OS Supervisor Services and Macro Instructions* are stated. HIARCHY options and those used only by OS multi-tasking systems are ignored by CMS. Validity checking is not performed within the simulation routines. The entry point name in LINK, XCTL, and LOAD (SVC 6, 7, 8) must be a member name or alias in a LOADLIB directory or in a TXTLIB directory unless the COMPSWT is set to on. If the COMPSWT is on, SVC 6, 7, and 8 must specify a module name. This switch is turned on and off by using the COMPSWT macro. See the *VM/SP CMS Command Reference* for descriptions of all CMS user macros.

Macro-SVC No. Differences in Implementation

XDAP-SVC 0 The TYPE option must be R or W; the V, I, and K options are not supported. The BLKREF-ADDR must point to an item number acquired by a NOTE macro. Other options associated with V, I, or K are not supported.

EXCP-SVC 0 The EXCP macro is supported by CMS. The EXCP macro executes graphic channel programs for graphic access method (GAM).

WAIT-SVC 1 All options of WAIT are supported. The WAIT routine waits for the completion bit to be set in the specified ECBs.

POST-SVC 2 All options of POST are supported. POST sets a completion code and a completion bit in the specified ECB.

EXIT/RETURN-SVC 3

Depending upon whether this is an exit or return from a linked or an attached routine, SVC 3 processing does the following: posts ECB, executes end of task routines, releases phase storage, unchains and frees latest request block, and restores registers. Do not use EXIT/RETURN to exit from an explicitly LOADed phase. If EXIT/RETURN is used for this purpose, CMS issues abend code A0A.

GETMAIN-SVC 4

All options of GETMAIN are supported except SP, BNDRY=, HIARCHY, LC, and LU. SP, BNDRY=, and HIARCHY are ignored by CMS. LC and LU result in abnormal termination if used. GETMAIN gets blocks of free storage.

FREEMAIN-SVC 5

All options of FREEMAIN are supported except SP and L. SP is ignored by CMS, and L results in abnormal termination if used. FREEMAIN frees blocks of storage acquired by GETMAIN.

GETPOOL/FREEPOOL

All the options of GETPOOL and FREEPOOL are supported. GETPOOL constructs a buffer pool and stores the address of a buffer pool control block in the DCB.

FREEPOOL frees a buffer pool constructed by GETPOOL.

LINK-SVC 6 The DCB and HIARCHY options are ignored by CMS. All other options of LINK are supported. LINK loads the specified program into storage (if necessary) and passes control to the specified entry point.

XCTL-SVC 7 The DCB and HIARCHY options are ignored by CMS. All other options of XCTL are supported. XCTL loads the specified program into storage (if necessary) and passes control to the specified entry point.

LOAD-SVC 8 The DCB and HIARCHY options are ignored by CMS. All other options of LOAD are supported. LOAD loads the specified program into storage (if necessary) and returns the address of the specified entry point in register 0. If loading a subroutine is required when SVC 8 is issued, CMS searches directories for a TXTLIB member containing the entry point or for a TEXT file with a matching filename. An entry name in an unloaded TEXT file will not be found unless the filename matches the entry name. After the subroutine is loaded, CMS tries to resolve external references within the subroutine, and may return another entry point address. To insure a correct address in register 0, the user should bring such subroutines into storage either by the CMS LOAD/INCLUDE commands or by a VCON in the user program.

DELETE-SVC 9 All the options of DELETE are supported. DELETE decreases the use count by one and, if the result is zero, frees the corresponding virtual storage. Code 4 is returned in register 15 if the phase is not found.

GETMAIN/FREEMAIN-SVC 10

All the options of GETMAIN and FREEMAIN are supported except SP and HIARCHY, which are ignored by CMS.

TIME-SVC 11 CMS supports only the DEC, BIN, TU, and MIC parameters of the TIME macro instruction. TIME returns the time of day to the calling program. However, the time value that CMS returns is only accurate to the nearest second and is converted to the proper unit.

ABEND-SVC 13 The completion code parameter is supported. The DUMP parameter is not. If a STAE request is outstanding, control is given to the proper STAE routine. If a STAE routine is not outstanding, a message indicating that an abend has occurred is printed on the terminal along with the completion code.

SPIE-SVC 14 All the options of SPIE are supported. The SPIE routine specifies interruption exit routines and program interruption types that cause the exit routine to receive control.

RESTORE-SVC 17 The RESTORE routine in CMS is a NOP. It returns control to the user.

BLDL-SVC 18 BLDL is an effective NOP for LINKLIBs and JOBLIBs. For TXTLIBs and MACLIBs, item numbers are filled in the TTR field of the BLDL list. The K, Z, and user data fields, as described in *OS/VS Data Management Macro Instructions,* are set to zeroes. The "alias" bit of the C field is supported, and the remaining bits in the C field are set to zero.

FIND-SVC 18 All the options of FIND are supported. FIND sets the read/write pointer to the item number of the specified member.

STOW-SVC 21 All the options of STOW are supported. The "alias" bit is supported, but the user data field is not stored in the MACLIB directory since CMS MACLIBs do not contain user data fields.

When using the STOW macro's ADD directory function without closing and reopening the data set after each new member is added, the CLOSE macro must be issued within each multiple of 256 new members. The existing number of entries does not need to be known before the ADD function is started.

OPEN/OPENJ-SVC 19/22 All the options of OPEN and OPENJ are supported except for the DISP, EXTEND, and RDBACK options, which are ignored. OPEN creates a CMSCB (if necessary), completes the DCB, and merges necessary fields of the DCB and CMSCB.

CLOSE/TCLOSE(CLOSE TYPE=T)-SVC 20/23 All the options of CLOSE and TCLOSE are supported except for the DISP option, which is ignored. The DCB is restored to its condition before OPEN. If the device type is disk, the file is closed. If the device type is tape, the REREAD option is treated as a REWIND. For TCLOSE, the REREAD option is REWIND, followed by a forward space file for tapes with standard labels.

DEVTYPE-SVC 24 With the exception of the RPS option, which CMS ignores, CMS accepts all options of the DEVTYPE macro instruction. In supporting this macro instruction, CMS groups all devices of a particular type into the same class.

For example, all printers are grouped into the printer class, all tape drives into the tape drive class, and so forth. In response to the DEVTYPE macro instruction, CMS provides the same device characteristics for all devices in a particular class. Thus, all devices in a particular class appear to be the same device type.

The device type characteristics CMS returns for each class are:

Class	**Device Characteristics**
Printer	1403
Virtual reader	2540
Console	1052
Tape drive	2400 (9 track)
DASD	2314
Virtual punch	2540
DUMMY	none
unassigned	2314

TRKBAL-SVC 25

The TRKBAL routine in CMS is a NOP. It returns control to the user.

FEOV-SVC 31 Control is returned to CMS with an error code of 4 in register 15.

WTO/WTOR-SVC 35

All options of WTO and WTOR are supported except those options concerned with multiple console support. WTO displays a message at the operator's console. WTOR displays a message at the operator's console, waits for a reply, moves the reply to the specified area, sets a completion bit in the specified ECB, and returns. There is no check made to determine if the operator provides a reply that is too long. The reply length parameter of the WTOR macro instruction specifies the maximum length of the reply. The WTOR macro instruction reads only this amount of data.

EXTRACT-SVC 40

The EXTRACT routine in CMS is essentially a NOP. The user-provided answer area is set to zeroes and control is returned to the user with a return code of 4 in register 15.

IDENTIFY-SVC 41

The IDENTIFY routine in CMS adds a REQUEST block to the load request chain for the requested name and address.

ATTACH-SVC 42

All the options of ATTACH are supported in CMS as in OS PCP. The following options are ignored by CMS: DCB, LPMOD, DPMOD, HIARCHY, GSPV, GSPL, SHSPV, SHSPL, SZERO, PURGE, ASYNCH, and TASKLIB. ATTACH passes control to the routine specified, fills in an ECB completion bit if an ECB is specified, passes control to an exit routine if one is specified, and returns control to the instruction following the ATTACH.

Since CMS is not a multitasking system, a phase requested by the ATTACH macro must return to CMS.

CHAP-SVC 44 The CHAP routine in CMS is a NOP. It returns control to the user.

TTIMER-SVC 46 All the options of TTIMER are supported.

STIMER-SVC 47 All options of STIMER are supported except for TASK and WAIT. The TASK option is treated as if the REAL option had been specified, and the WAIT option is treated as a NOP; it returns control to the user. The maximum time interval allowed is X'7FFFFF00' timer units (or 15 hours, 32 minutes, and 4 seconds in decimal). If the time interval is greater than the maximum, it is set to the maximum.

Note: If running in the CMSBATCH environment, issuing the STIMER or TTIMER macro affects the CMSBATCH time limit. Depending on the frequency, number, and duration of STIMERs and/or TTIMERs issued, the CMSBATCH limit may never expire.

DEQ-SVC 48 The DEQ routine in CMS is a NOP. It returns control to the user.

SNAP-SVC 51 Except for SDATA, PDATA, and DCB, all options of the SNAP macro are processed normally. SDATA and PDATA are ignored. Processing for the DCB option is as follows. The DBC address specified with SNAP is used to verify that the file associated with the DCB is open. If it is not open, control is returned to the caller with a return code of 4. If the file is open, then storage is dumped (unless the FCB indicates a DUMMY device type). SNAP always dumps output to the printer. The dump contains the PSW, the registers, and the storage specified.

ENQ-SVC 56 The ENQ routine in CMS is a NOP. It returns control to the user.

FREEDBUF-SVC 57
All the options of FREEDBUF are supported. FREEDBUF returns a buffer to the buffer pool assigned to the specified DCB.

STAE-SVC 60 All the options of STAE are supported except for the XCTL option, which is set to XCTL=YES; the PURGE option, which is set to HALT; and the ASYNCH option, which is set to NO. STAE creates, overlays, or cancels a STAE control block as requested. STAE retry is not supported.

DETACH-SVC 62

The DETACH routine in CMS is a NOP. It returns control to the user.

CHKPT-SVC 63 The CHKPT routine is a NOP. It returns control to the user.

RDJFCB-SVC 64 All the options of RDJFCB are supported. RDJFCB causes a job file control block (JFCB) to be read from a CMS control block (CMSCB) into real storage for each data control block specified. FILEDEF commands create CMSCBs.

Additional information regarding CMS OS Simulation of RDJFCB follows:

- The DCBs specified in the RDJFCB PARAMETER LIST are processed sequentially as they appear in the parameter list.

- On return to the caller, a return code of zero is always placed in register 15. If an abend occurs, control is not returned to the caller.

- Abend 240 occurs if zero is specified as the address of the area into which the JFCB is to be placed.

- Abend 240 occurs if a JFCB EXIT LIST ENTRY (Entry type X'07') is not present in the DCB EXIT LIST for any one of the DCBs specified in the RDJFCB PARAMETER LIST.

- If a DCB is encountered in the parameter list with zero specified as the DCB EXIT LIST (‘EXLST’) address, the RDJFCB immediately returns with return code zero in register 15. Except for this situation, all of the DCBs specified in the RDJFCB PARAMETER LIST are processed, unless an abend occurs.

- For a DCB that is not open, a search is done for the corresponding FILEDEF or DLBL. If one is not found, a test is done to determine if a file exists with a filename of ‘FILE’, a filetype of the DDNAME from DCB, and a filemode of ‘A1’. If such a file does exist, then X'40' is placed in the JFCB at displacement X'57' (FLAG ‘JFCOLD IN FIELD ‘JFCBIND2’). If such a file does not exist then X'C0' (FLAG 'JFCNEW') will be in field 'JFCBIND2'.

- For a file that is not open, but for which a DLBL has been specified, X'08' is placed in the JFCB at displacement X'63' (field 'JFCDSORG' byte 2) to indicate that it is a VSAM file.

SYNADAF-SVC 68

All the options of SYNADAF are supported. SYNADAF analyzes an I/O error and creates an error message in a work buffer.

SYNADRLS-SVC 68

All the options of SYNADRLS are supported. SYNADRLS frees the work area acquired by SYNAD and deletes the work area from the save area chain.

BSP-SVC 69

All the options of BSP are supported. BSP decrements the item pointer by one block.

TGET/TPUT-SVC 93

TGET and TPUT operate as if EDIT and WAIT were coded. TGET reads a terminal line. TPUT writes a terminal line.

TCLEARQ-SVC 94

TCLEARQ in CMS clears the input terminal queue and returns control to the user.

STAX-SVC 96

The only option of STAX that is supported is EXIT ADDRESS. STAX updates a queue of CMTAXEs each of which defines an attention exit level.

PGRLSE-SVC 112

Release all complete pages (4K bytes) associated with the area of storage specified.

CALL

The CALL macro is supported by CMS. The CALL macro transfers control to a control section at a specified entry.

NOTE

All the options of NOTE are supported. NOTE returns the item number of the last block read or written.

POINT

All the options of POINT are supported. POINT causes the control program to start processing the next read or write operation at the specified item number. The TTR field in the block address is used as an item number.

CHECK

All the options of CHECK are supported. CHECK tests the I/O operation for errors and exceptional conditions.

DCB

The following fields of a DCB may be specified relative to the particular access method indicated:

Operand	BDAM	BPAM	BSAM	QSAM
BFALN	F,D	F,D	F,D	F,D
BLKSIZE	n(number)	n	n	n
BUFCB	a(address)	a	a	a
BUFL	n	n	n	n
BUFNO	n	n	n	n
DDNAME	s(symbol)	s	s	s
DSORG	DA	PO	PS	PS
EODAD	-	a	a	a
EXLST	a	a	a	a
KEYLEN[1]	n	-	n	-
LIMCT	n	-	-	-
LRECL	-	n	n	n
MACRF	R,W	R,W	R,W,P	G,P,L,M
OPTCD	A,E,F,R	-	J	J
RECFM	F,V,U	F,V,U,	F,V,B,S,A,M,U	F,V,B,U,A,M,S
SYNAD	a	a	a	a
NCP	-	n	n	-

Access Method Support

An access method governs the manipulation of data. To facilitate the execution of OS code under CMS, the processing program must see data as OS would present it. For instance, when the processors expect an access method to acquire input source cards sequentially, CMS invokes specially written routines that simulate the OS sequential access method and pass data to the processors in the format that the OS access methods would have produced. Therefore, data appears in storage as if it had been manipulated using an OS access method. For example, block descriptor words (BDW), buffer pool management, and variable records are updated in storage as if an OS access method had processed the data. The actual writing to and reading from the I/O device is handled by CMS file management. Note that the character string X'61FFFF61' is interpreted by CMS as an end of file indicator.

The essential work of the volume table of contents (VTOC) and the data set control block (DSCB) is done in CMS by a master file directory (MFD) that updates the disk contents and a file status table (FST). A MFD updates the disk contents, and a FST describes each data file. All disks are formatted in physical blocks of 512, 800, 1K, 2K, or 4K bytes.

CMS continues to update the OS format, within its own format, on the auxiliary device for files whose filemode number is 4. That is, the block and record descriptor words (BDW and RDW) are written along with the data. If a data set consists of blocked records, the data is written to and read from the I/O device in physical blocks rather than logical records. CMS also simulates the specific methods of manipulating data sets.

[1] If an input data set is not a BDAM data set, zero is the only value that should be specified for KEYLEN. This applies to the user exit lines as well as to the DCB macro instruction.

When the OPEN macro instruction is executed, the CMS simulation of the OS OPEN routine initializes the data control block (DCB). The DCB fields are filled in with information from the DCB macro instruction, the information specified on the FILEDEF command, or, if the data set already exists, the data set label. However, if more than one source specifies information for a particular field, only one source is used.

The DCB fields are filled in this order:

1. The DCB macro instruction in your program

2. The fields you had specified on the FILEDEF command

3. The data set label if the data set already exists.

The DCB macro instruction takes precedence over the FILEDEF and the data set label. This FILEDEF takes precedence over the data set label. Data set label information from an existing CMS file is used only when the OPEN is for input or update, otherwise, the OPEN routine erases the existing file.

You can modify any DCB field either before the data set is opened or through a data control block open exit. CMS supports only the data control block exit of the EXIT LIST (EXLST) options.

When the data set is closed, the DCB is restored to its original condition. Fields that were merged in at OPEN time from the FILEDEF and the data set label are cleared.

To accomplish this simulation, CMS supports certain essential macros for the following access methods:

BDAM (direct) - identifying a record by a key or by its relative position within the data set.

BPAM (partitioned) - seeking a named member within data set.

Note: Two BPAM files with the same filetype cannot be updated at the same time. The reason for this restriction is that DMSSVT uses the filetype of the file in the DCB for the filetype of the temporary BPAM directory file. Therefore, when opening more than one BPAM file at the same time could result in MSDMSSOP036E Error Code 8.

BSAM/QSAM (sequential) - accessing a record in a sequence in relation to preceding or following records.

VSAM (direct or sequential) - accessing a record sequentially or directly by key or address.

Note: CMS support of OS VSAM files is based on VSE/VSAM. Therefore, the OS user is restricted to those functions available under VSE/VSAM. See "CMS Support for OS and VSE VSAM Functions" on page 54 for details.

CMS also updates those portions of the OS control blocks that are needed by the OS simulation routines to support a program during execution. Most of the simulated supervisory OS control blocks are contained in the following two CMS control blocks:

CMSCVT simulates the communication vector table. Location 16 contains the address of the CVT control section.

CMSCB is allocated from system free storage whenever a FILEDEF command or an OPEN (SVC 19) is issued for a data set. The CMS control block consists of a file control block (FCB) for the data file and partial simulation of the job file control block (JFCB), input/output block (IOB), and data extent block (DEB).

The data control block (DCB) and the data event control block (DECB) are used by the access method simulation routines of CMS.

Note: The results may be unpredictable if two DCBs access the same data set at the same time.

The GET and PUT macros are not supported for use with spanned records except in GET locate mode. READ, WRITE and GET (in locate mode) are supported for spanned records, provided the filemode number is 4 and the data set is in physical sequential format.

GET (QSAM)
All the QSAM options of GET are supported. Substitute mode is handled the same as move mode. When the DCBRECFM is FB, the filemode number is 4, the last block is a short block, and an EOF indicator (X'61FFFF61') must be present in the last block after the last record. Issue an explicit CLOSE prior to returning to CMS to obtain the last record when LOCATE mode is used with PUT.

GET (QISAM)
QISAM is not supported in CMS.

PUT (QSAM)
All the QSAM options of PUT are supported. Substitute mode is handled the same as move mode. If the DCBRECFM is FB, the filemode number is 4, and the last block is a short block. An EOF indicator is written in the last block after the last record. When LOCATE mode is used with PUT, issue an explicit CLOSE prior to returning to CMS to obtain the last record.

PUT (QISAM)
QISAM is not supported in CMS.

PUTX
PUTX support is provided only for data sets opened for QSAM-UPDATE with simple buffering.

READ/WRITE (BISAM)
BISAM is not supported in CMS.

READ/WRITE (BSAM and BPAM)
All the BSAM and BPAM options of READ and WRITE are supported except for the SB option (read backwards).

READ (Offset Read of Keyed BDAM data set)
This type of READ is not supported because it is used only for spanned records.

READ/WRITE (BDAM)
All the BDAM and BSAM (create) options of READ and WRITE are supported except for the R and RU options.

When an input or output error occurs, do not depend on OS sense bytes. An error code is supplied by CMS in the ECB in place of the sense bytes. These error codes differ for various types of devices and their meaning can be found in *VM/SP System Messages and Codes*, under DMS message 120S.

Note: If OPTCD J is specified in the FILEDEF command, the proper flag is set in the JFCOPTCD byte of the FCBSECT (simulated OS control block). During simulation of the OS OPEN macro, the FILEDEF value is merged into DCBOPTCD. After DCBOPTCD is set, the first data byte of output lines presented to the PUT (QSAM) and WRITE (BSAM) macros is interpreted as a table reference character (TRC) byte. CP uses the TRC byte to select translate tables when printing on a 3800. The translate table determines the font type at real print time. If the virtual printer is not a 3800, the TRC byte is stripped off and the line is printed in the usual manner.

BDAM Restrictions

The four methods of accessing BDAM records are:

1. Relative Block RRR
2. Relative Track TTR
3. Relative Track and Key TTK
4. Actual Address MBBCCHHR.

The restrictions on these access methods are as follows:

- Only the BDAM identifiers underlined above can be used to refer to records since the CMS simulation of BDAM files uses a three-byte record identifier on 512, 1K, 2K, and 4K format CMS minidisks. For 800-byte disks, only the last two identifiers are used.

- CMS BDAM files are always created with 255 records on the first logical track and 256 records on all other logical tracks, regardless of the block size. If BDAM methods 2, 3, or 4 are used and the RECFM is U or V, the BDAM user must either write 255 records on the first track and 256 records on every track thereafter, or the BDAM user must not update the track indicator until a NO SPACE FOUND message is returned on a write. For method 3 (WRITE ADD), this message occurs when no more dummy records can be found on a WRITE request. For methods 2 and 4, this does not occur and the track indicator is updated only when the record indicator reaches 256 and overflows into the track indicator.

- The user must create variable length BDAM files (in PL/I they are regional 3 files) entirely under CMS. Also specify, on the XTENT option of the FILEDEF command, the exact number of records to be written. When reading variable length BDAM files, the XTENT and KEYLEN information specified for the file must duplicate the information specified when the file was created. CMS does not support WRITE ADD of variable length BDAM files; that is, the user cannot add additional records to the end of an already existing variable length BDAM file.

- Two files of the same filetype, both using keys, cannot be open at the same time. If a program that is updating keys does not close the file it is updating for some reason, such as a system failure or another IPL operation, the original keys for files that are not fixed format are saved in a temporary file with the same filetype and a filename of $KEYSAVE. To finish the update, run the program again.

- Variable length BDAM files must be created under CMS in their entirety, with the XTENT option of FILEDEF specifying the exact number of records to be written. When reading variable BDAM files, the XTENT and key length information specified must duplicate what was created at file creation time. CMS does not support adding variable length records to BDAM files.

- Once a file is created using keys, additions to the file must not be made without using keys and specifying the original length.

- Note that there is limited support from the CMS file system for BDAM created files (sparse). Sparse files are manipulated with CMS commands but are not treated as sparse files by most CMS commands. The number of records in the FST is treated as a valid record number.

- The number of records in the data set extent must be specified using the FILEDEF command. The default size is 50 records.

- The minimum LRECL for a CMS BDAM file with keys is eight bytes.

Reading OS Data Sets and DOS Files Using OS Macros

CMS users can read OS sequential and partitioned data sets that reside on OS disks. The CMS MOVEFILE command can be used to manipulate those data sets, and the OS QSAM, BPAM, and BSAM macros can be executed under CMS to read them.

The CMS MOVEFILE command and the same OS macros can also be used to manipulate and read DOS sequential files that reside on DOS disks. OS macros, however, can only be used to read sequential files from DOS formatted CKD disks. OS macros are not supported for reading sequential files on DOS formatted FB-512 disks.

The following OS Release 20.0 BSAM, BPAM, and QSAM macros can be used with CMS to read OS data sets and DOS files:

BLDL
BSP
CHECK
CLOSE
DEQ
DEVTYPE
ENQ
FIND
GET
NOTE
POINT
POST
RDJFCB
READ
SYNADAF
SYNADRLS
WAIT

CMS supports the following disk formats for the OS and OS/VS sequential and partitioned access methods:

- Split cylinders
- User labels
- Track overflow
- Alternate tracks.

As in OS, the CMS support of the BSP macro produces a return code of 4 when trying to backspace over a tape mark or when a beginning of an extent is found on an OS data set or a VSE file. If the data set or data file contains split cylinders, an attempt to backspace within an extent, resulting in a cylinder switch, also produces a return code of 4. When a data set has been allocated or updated by OS on an OS disk, an OS CLOSE must be issued before CMS can read or move it. The CLOSE marks the end-of-file (EOF) and updates the DS1LSTAR field of the Format 1 DSCB. If the CLOSE is not issued, CMS may read or move residual data that remains beyond the intended end of the file.

The ACCESS Command

Before CMS can read an OS data set or VSE file that resides on a non-CMS disk, you must issue the CMS ACCESS command to make the disk available to CMS.

The format of the ACCESS command is:

ACCESS	*cuu mode* [*/ext*]

For more details, see the *CMS Command Reference.* You must not specify options or file identification when accessing an OS or DOS disk.

The FILEDEF Command

You then issue the FILEDEF command to assign a CMS file identification to the OS data set or VSE file so that CMS can read it.

The format of the FILEDEF command used for this purpose is:

```
FIledef    ┌                                                                       ┐
           │              ┌ DISK  ┌ fn    ft     ┐ ┌ fm ┐                      ┐   │
           │              │       └ FILE  ddname ┘ └ A1 ┘                      │   │
           │              │ or                                                 │   │
           │ ┌ ddname ┐   │                                                    │   │
           │ │ nn     │   │ ┌┌┌ DISK  fn    ft     ┐ ┌ fm ┐┐ ┌ DSN ?          ┐┐  │   │
           │ └ *      ┘  <│ │││       FILE  ddname │ │ A1 ││ │ DSN qual1 qual2 ...││ >│   │
           │              │ └└└                    ┘ └    ┘┘ └ DSN qual1.qual2 ...┘┘  │   │
           │              │                                                    │   │
           │              │ DUMMY                                              │   │
           │              └                                                    ┘   │
           │                                                                       │
           │  Related Options:  ┌ MEMBER  membername ┐                             │
           │                    └ CONCAT             ┘                             │
           └                                                                       ┘
```

If you are issuing a FILEDEF for a VSE file, note that the OS program that will use the VSE file must have a DCB for it. For *ddname* in the FILEDEF command line, use the *ddname* in that DCB. With the DSN operand, enter the fileid of the VSE file.

Sometimes, CMS issues the FILEDEF command for you. Although the CMS MOVEFILE command, the supported CMS licensed program interfaces, and the CMS OPEN routine each issue a default FILEDEF, you should issue the FILEDEF command yourself to ensure the appropriate file is defined.

After you have issued the ACCESS and FILEDEF commands for an OS sequential data set, OS partitioned data set, or VSE sequential file, CMS commands (such as ASSEMBLE and STATE) can refer to the OS data set or VSE file just as if it were a CMS file.

Several other CMS commands can be used with OS data sets and DOS files that do not reside on CMS disks. See the *VM/SP CMS Command Reference* for a complete description of the CMS ACCESS, FILEDEF, LISTDS, LKED, MOVEFILE, OSRUN, QUERY, RELEASE, and STATE commands.

For restrictions on reading OS data sets and DOS files under CMS, see the *VM/SP CMS for System Programming.*

The CMS FILEDEF command allows you to specify the I/O device and the file characteristics to be used by a program at execution time. In conjunction with the OS simulation scheme, FILEDEF simulates the functions of the data definition JCL statement.

FILEDEF may be used only with programs using OS macros and functions. For example:

```
filedef  file1  disk  proga  data  a1
```

After issuing this command, your program referring to FILE1 would access PROGA DATA on your A-disk.

If you wished to supply data from your terminal for FILE1, you could issue the command:

```
filedef  file1  terminal
```

and enter the data for your program without recompiling.

```
fi  tapein  tap2  (recfm fb lrecl 50 block 100 9track den 800)
```

After issuing this command, programs referring to TAPEIN access a tape at virtual address 182. (Each tape unit in the CMS environment has a symbolic name associated with it.) The tape must have been previously attached to the virtual machine by the VM/SP operator.

To maintain OS compatibility in the EOV2/EOF2 label, you must specify LRECL in the output FILEDEF.

The AUXPROC Option of the FILEDEF Command:

The AUXPROC option can only be used by a program call to FILEDEF and not from the terminal. The CMS language interface programs use this feature for special I/O handling of certain (utility) data sets.

The AUXPROC option, followed by a fullword address of an auxiliary processing routine, allows that routine to receive control from DMSSEB before any device I/O is performed. At the completion of its processing, the auxiliary routine returns control to DMSSEB signaling whether or not I/O has been performed. If it has not been done, DMSSEB performs the appropriate device I/O.

When control is received from DMSSEB, the general purpose registers contain the following information:

GPR2 = data control block (DCB address)

GPR3 = base register for DMSSEB

GPR8 = CMS OPSECT address

GPR11 = file control block (FCB) address

GPR14 = return address in DMSSEB

GPR15 = auxiliary processing routine address

all other registers = work registers

The auxiliary processing routine must provide a save area to save the general registers. This routine must also perform the save operation. DMSSEB does not provide the address of a save area in general register 13, as is usually the case. When control returns to DMSSEB, the general registers must be restored to their original values. Control is returned to DMSSEB by branching to the address contained in general register 14.

GPR15 is used by the auxiliary processing routine to inform DMSSEB of the action that has been or should be taken with the data block as follows:

Register	Action
GPR15=0	No I/O performed by AUXPROC routine. DMSSEB performs I/O.
GPR15<0	I/O performed by AUXPROC routine and error was encountered. DMSSEB takes error action.
GPR15>0	I/O performed by AUXPROC routine with residual count in GPR15. DMSSEB returns normally.
GPR15=64K	I/O performed by AUXPROC routine with zero residual count.

CMS QSAM Tape End-of-Volume Exit

A program working with CMS simulation of OS QSAM can set up an exit that could be entered on the end-of-volume condition on IBM standard label tapes. This exit receives control after the trailer labels have been processed and the tape has been rewound and unloaded, provided that OS simulation is for something other than standard labels. For standard labels multivolume standard label processing is done before termination. This exit should not be confused with the OS DCB end-of-volume exit. The OS DCB end-of-volume exit continues to be unsupported.

TEOVEXIT Macro

Use the TEOVEXIT macro instruction to set up and clear a CMS tape end-of-volume exit.

The four formats of the TEOVEXIT macro instruction are:

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

Standard Format

The standard format of the TEOVEXIT macro is:

[*label*]	**TEOVEXIT**	{ **SET,DDNAME** = {*'ddname'* | *addr*} **,EXIT** = *addr* **,RETINFO** = *addr* [**,ERROR** = *addr*] **CLR,DDNAME** = {*'ddname'* | *addr*} [**,ERROR** = *addr*] }

where:

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

label
is an assembler program label.

SET
establishes an exit.

CLR
clears an exit.

DDNAME =
is the *ddname* the tape end-of-volume exit is being established for. *ddname* may be from 1 to 8 alphameric characters enclosed in quotes.

EXIT =

label is an assembler program label that is the address of the program's end-of-volume processing routine.

(R*n*) is a general register. Its value is the address of the program's end-of-volume processing routine.

This routine receives control after trailer labels have been processed and the tape has been rewound and unloaded. This routine receives control with the same PSW key as the call to CMS QSAM. The registers passed to the exit are the same as they were at the call to QSAM except: register 0 points to the DCB: register 1 points to the FCB; register 14 contains the address the routine branches to upon completion. If the exit does not return control to the address in register 14, future options are unpredictable for that file. Register 15 contains the address of the user exit routine.

(This attribute is required for SET. If the EXIT attribute is specified on CLR, it is ignored. No MNOTE is issued.)

Note: When control is returned to the program that issued the QSAM call, the registers are unaffected by changes to registers in the end-of-volume exit.

RETINFO=

label is an assembler program label that is the address of a 20-byte halfword aligned area.

(R*n*) is a general register. Its value is the address of a 20-byte halfword aligned area.

The program must provide this 20 byte, halfword aligned area for return information.

(This attribute is required for SET. If the RETINFO attribute is specified on CLR, it is ignored. No MNOTE is issued.)

ERROR=

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

(R*n*) is a general register. Its value is the address of the error routine.

The error routine receives control if an error is found. If this parameter is not specified and an error occurs, control returns to the next sequential instruction in the calling program.

List Format (MF=L)

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

[*label*]	**TEOVEXIT MF=L**	[**,DDNAME**=*'ddname'*] [**,EXIT**=*label*] [**,RETINFO**=*label*] **,SET** [**,DDNAME**=*'ddname'*] [**,EXIT**= *label*] **,CLR** [**,DDNAME**=*'ddname'*]

All parameters have the same meaning as the standard format with the following difference:

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

Note: When using the MF= parameter, all other parameters are optional. When the function is executed, however, a valid combination of parameters must have been specified by the LIST and EXECUTE formats of the macro.

Complex List Format (MF=(L,addr [,label]))

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

[*label*]	**TEOVEXIT MF=(L,** *addr* [*,label*]**)**	[[**,DDNAME=**{*'ddname'* | *addr*}] [**,EXIT=** *addr*] [**,RETINFO =** *addr*] **,SET** [**,DDNAME=**{*'ddname'* | *addr*}] [**,EXIT=** *addr*] [**,RETINFO =** *addr*] **,CLR**[**,DDNAME=**{*'ddname'* |*addr*}]]

All parameters have the same meaning as the standard format with the following difference:

MF=(L,*addr*[,*label*])
indicates that the parameter list is created in the area specified by *addr*. The address may represent an area within your program or an area of free storage obtained by a system service. You can determine the size of the parameter list by coding the *label* operand. The macro expansion equates *label* to the size of the parameter list. This format of the macro produces executable code to move the data into the parameter list specified by *addr*. However, it does not generate the instructions to invoke the function. If this version of the LIST format is used, it must be executed before any related invocation of the EXECUTE format.

Note: When using the MF= parameter, all other parameters are optional. When the function is executed, however, a valid combination of parameters must have been specified by the LIST and EXECUTE formats of the macro.

Execute Format (MF=(E,addr))

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

[*label*]	**TEOVEXIT MF=(E,** *addr***)**	[[**,DDNAME**={'*ddname*' | *addr*}] [**,EXIT**= *addr*] [**,RETINFO** = *addr*] [**,ERROR**= *addr*] **,SET** [**,DDNAME**= {'*ddname*' | *addr*}] [**,EXIT**= *addr*] [**,RETINFO** = *addr*] [**,ERROR**= *addr*] **,CLR** [**,DDNAME**= {'*ddname*' | *addr*}] [**,ERROR**= *addr*]]

All parameters have the same meaning as the standard format with the following difference:

MF=(E,*addr*)
indicates that instructions are generated to execute the TEOVEXIT function. *addr* represents the location of the parameter list. Information in the parameter list may be changed by specifying the appropriate operands on the macro.

Note: When using the MF= parameter, all other parameters are optional. When the function is executed, however, a valid combination of parameters must have been specified by the LIST and EXECUTE formats of the macro.

Restrictions

1. Tape end-of-volume exit only applies to CMS OS QSAM simulation.

2. Only IBM standard label tapes are supported. If other than standard labels are used, you receive a return code of 16 from TEOVEXIT.

3. The LEAVE option of the FILEDEF command is invalid. If it is used, you receive a return code of 20 from TEOVEXIT.

4. The NOEOV processing option of the FILEDEF command is invalid. If it is used, you receive a return code of 28 from TEOVEXIT.

5. You cannot read backwards. If it is attempted, the results are unpredictable.

6. The tape end-of-volume exit is not entered if either an OPEN or a CLOSE is in progress.

7. The exit must not issue I/O requests that might result in the tape end-of-volume exit being invoked. If it is attempted, the results are unpredictable.

8. The exit must not issue additional QSAM requests to the file. If it is attempted, the results are unpredictable.

9. The exit must not modify or clear the FCB of the file the end-of-volume condition was encountered on.

10. TEOVEXITs are cleared whenever a CLOSE or a CLOSE type T is issued for the file.

Return Codes

If any errors occur during the processing of the TEOVEXIT macro, register 15 contains the error return codes.

SET Function

Code	Meaning
0	End-of-volume exit is established for the specified DDNAME. This is the normal return.
4	The DDNAME specified is not found. (No FILEDEF was found with the given DDNAME.)
8	Device specified in the FILEDEF is not a tape device.
12	Tape identification is invalid. (Must be TAP0-TAPF.)
16	Tape label type is other than "SL"
20	"LEAVE" is specified in the FILEDEF (FCB).
24	Invalid PLIST.
28	"NOEOV" is specified in the FILEDEF (FCB).
32	Exit address or RETINFO address is zero.

CLR Function

Code	Meaning
0	End-of-volume exit is cleared for the specified DDNAME. This is the normal return. A return code of 0 may also indicate the end-of-volume exit was not in effect, but it was still cleared.
4	The DDNAME specified is not found. (No FILEDEF was found with the given DDNAME.)
24	Invalid PLIST.

Successful Completion

On successful completion of TEOVEXIT SET (register 15=0), the RETINFO attribute contains:

Word	**Meaning**
0	The symbolic tape number associated with the given DDNAME (character TAP0-TAPF)
1	The address of the FCB of the given DDNAME
2	RESERVED
3	RESERVED
4	RESERVED

Chapter 5. VSE Support Under CMS

CMS supports interactive program development for VSE. This includes creating, compiling, testing, debugging, and executing commercial application programs. The VSE programs can be executed in a CMS virtual machine or in a CMS Batch Facility virtual machine.

VSE files and libraries can be read under CMS. VSAM data sets can be read and written under CMS.

The CMS VSE environment (called CMS/DOS) provides many of the same facilities that are available in VSE. However, CMS/DOS supports only those facilities that are supported by a single (background) partition. The VSE facilities provided by CMS/DOS are:

- VSE linkage editor
- Fetch support
- VSE Supervisor and I/O macros
- VSE Supervisor control block support
- Transient area support
- VSE/VSAM macros.

This environment is entered each time the CMS SET DOS ON command is issued; VSAM functions are available in CMS/DOS only if the SET DOS ON (VSAM) command is issued. In the CMS/DOS environment, CMS supports many VSE facilities, but does not support OS simulation. When you no longer need VSE support under CMS, you issue the SET DOS OFF command and VSE facilities are no longer available.

CMS/DOS can execute programs that use the sequential access method (SAM) and virtual storage access method (VSAM), and CMS/DOS can access VSE libraries.

CMS/DOS cannot execute programs that have execution-time restrictions, such as programs that use sort exits, teleprocessing access methods, or multi-tasking. DOS/VS COBOL, DOS PL/I, DOS/VS RPG II and Assembler language programs are executable under CMS/DOS.

All of the CP and CMS online debugging and testing facilities (such as the CP ADSTOP and STORE commands and the CMS DEBUG environment) are supported in the CMS/DOS environment. Also, CP disk error recording and recovery is supported in CMS/DOS.

With its support of a CMS/DOS environment, CMS becomes an important tool for VSE application program development. Because CMS/DOS is a

VSE program development tool, it assumes that a VSE system exists, and uses it. The following sections describe what is supported and what is not.

CMS Support for OS and VSE VSAM Functions

CMS supports interactive program development for OS and VSE programs using VSE/VSAM. CMS supports VSAM macros for OS and VSE programs. The complete set of VSE/VSAM macros and options and a subset of OS/VSAM macros are supported for execution with Assembler language programs.

CMS also supports Access Method Services to manipulate OS and VSE VSAM and SAM data sets.

Under CMS, VSAM data sets can span up to 10 DASD volumes. CMS does not support VSAM data set sharing. However, CMS already supports the sharing of minidisks or full pack minidisks.

VSAM data sets created in CMS are not in the CMS file format. Therefore, CMS commands currently used to manipulate CMS files cannot be used for VSAM data sets that are read or written in CMS. A VSAM data set created in CMS has a file format that is compatible with OS and DOS VSAM data sets. Thus, a VSAM data set created in CMS can later be read or updated by OS or DOS. This compatibility with OS is limited to VSAM data sets created with physical record sizes of 512, 1K, 2K, and 4K bytes. For further information on compatibility between OS/VS VSAM and VSE/VSAM, please refer to *VSE/VSAM General Information.*

Because VSAM data sets in CMS are not a part of the CMS file system, CMS file size, record length, and minidisk size restrictions do not a apply. The VSAM data sets are manipulated with Access Method Services programs executed under CMS instead of with the CMS file system commands. Also, all VSAM minidisks and full packs used in CMS must be initialized by the Device Support Facility (DSF); the CMS FORMAT command must not be used.

CMS supports VSAM control blocks with the GENCB, MODCB, TESTCB, and SHOWCB macros.

In its support of VSAM data sets, CMS uses rotational position sensing (RPS) wherever possible. CMS does not use RPS for 2314/2319 devices or for 3340 devices that do not have the feature.

Hardware Devices Supported

CMS support of VSAM data sets is based on VSE/VSAM. The disks used for VSAM data sets in CMS are:

- IBM 2314 Direct Access Storage Facility

- IBM 2319 Disk Storage
- IBM 3310 Direct Access Storage
- IBM 3330 Disk Storage Models 1 and 2
- IBM 3330 Disk Storage, Model 11
- IBM 3340 Direct Access Storage Facility
- IBM 3344 Direct Access Storage
- IBM 3350 Direct Access Storage
- IBM 3370 Direct Access Storage, Models A1, A2, B1, and B2
- IBM 3375 Direct Access Storage
- IBM 3380 Direct Access Storage.

CMS disk files used as input to or output from Access Method Services may reside on any disk supported by CMS.

Part 2: Method of Operation and Program Organization

This part contains the following information:

- Initialization of the CMS virtual machine environment
- Processing and executing CMS files
- Processing commands that manipulate the file system
- Managing the CMS file system
- Handling I/O operations
- Handling interruptions
- Managing CMS storage
- Simulating non-CMS operating environments
- Performing miscellaneous CMS functions.

The CMS description is in two parts. The first part contains figures showing the functional organization of CMS. The second part contains general information about the internal structure of CMS programs and their interaction with one another.

CMS program organization is in two figures. Figure 7 on page 58 is an overview of the functional areas of CMS. Each block is numbered and corresponds to a more detailed outline of the function found in Figure 8 on page 59.

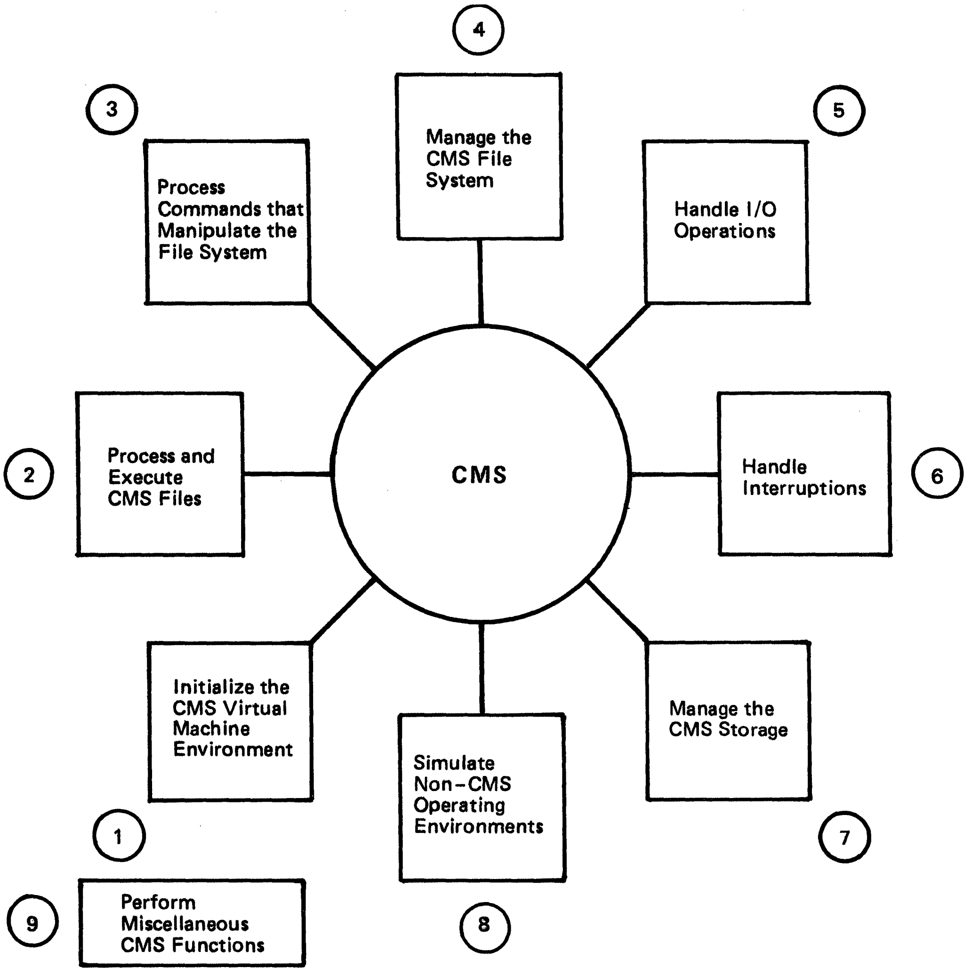

4
Manage the CMS File System
3
Process Commands that Manipulate the File System
5
Handle I/O Operations
2
Process and Execute CMS Files
CMS
Handle Interruptions
6
Initialize the CMS Virtual Machine Environment
1
Simulate Non-CMS Operating Environments
Manage the CMS Storage
7
9
Perform Miscellaneous CMS Functions
8

Figure 7. An Overview of the Functional Areas of CMS

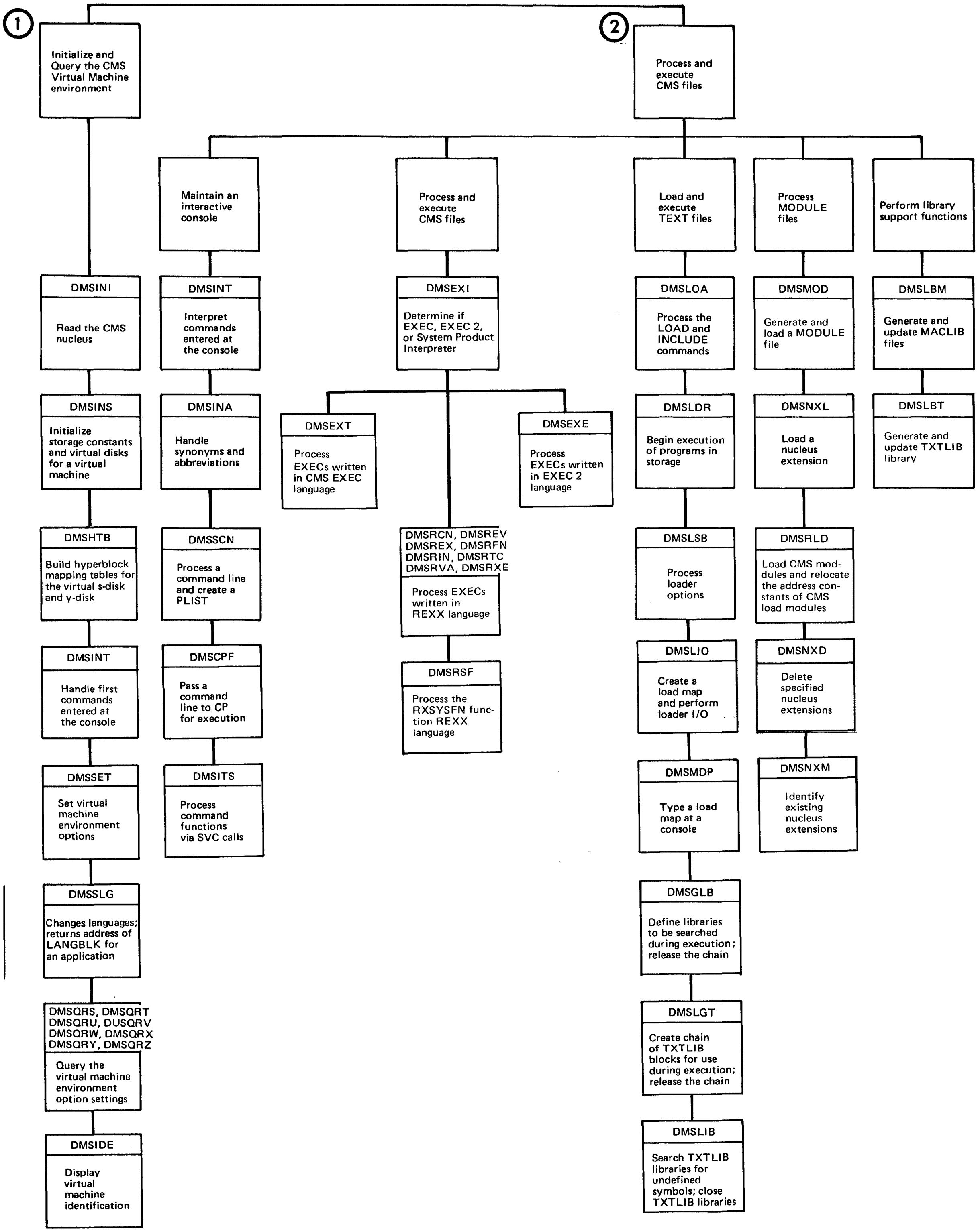

1
Initialize and Query the CMS Virtual Machine environment
2
Process and execute CMS files
Maintain an interactive console
Process and execute CMS files
Load and execute TEXT files
Process MODULE files
Perform library support functions
DMSINI
Read the CMS nucleus
DMSINT
Interpret commands entered at the console
DMSEXI
Determine if EXEC, EXEC 2, or System Product Interpreter
DMSLOA
Process the LOAD and INCLUDE commands
DMSMOD
Generate and load a MODULE file
DMSLBM
Generate and update MACLIB files
DMSINS
Initialize storage constants and virtual disks for a virtual machine
DMSINA
Handle synonyms and abbreviations
DMSEXT
Process EXECs written in CMS EXEC language
DMSEXE
Process EXECs written in EXEC 2 language
DMSLDR
Begin execution of programs in storage
DMSNXL
Load a nucleus extension
DMSLBT
Generate and update TXTLIB library
DMSHTB
Build hyperblock mapping tables for the virtual s-disk and y-disk
DMSSCN
Process a command line and create a PLIST
DMSRCN, DMSREV
DMSREX, DMSRFN
DMSRIN, DMSRTC
DMSRVA, DMSRXE
Process EXECs written in REXX language
DMSLSB
Process loader options
DMSRLD
Load CMS modules and relocate the address constants of CMS load modules
DMSINT
Handle first commands entered at the console
DMSCPF
Pass a command line to CP for execution
DMSRSF
Process the RXSYSFN function REXX language
DMSLIO
Create a load map and perform loader I/O
DMSNXD
Delete specified nucleus extensions
DMSSET
Set virtual machine environment options
DMSITS
Process command functions via SVC calls
DMSMDP
Type a load map at a console
DMSNXM
Identify existing nucleus extensions
DMSSLG
Changes languages; returns address of LANGBLK for an application
DMSGLB
Define libraries to be searched during execution; release the chain
DMSQRS, DMSQRT
DMSQRU, DUSQRV
DMSQRW, DMSQRX
DMSQRY, DMSQRZ
Query the virtual machine environment option settings
DMSLGT
Create chain of TXTLIB blocks for use during execution; release the chain
DMSIDE
Display virtual machine identification
DMSLIB
Search TXTLIB libraries for undefined symbols; close TXTLIB libraries

Figure 8 (Part 1 of 5). Details of CMS System Functions

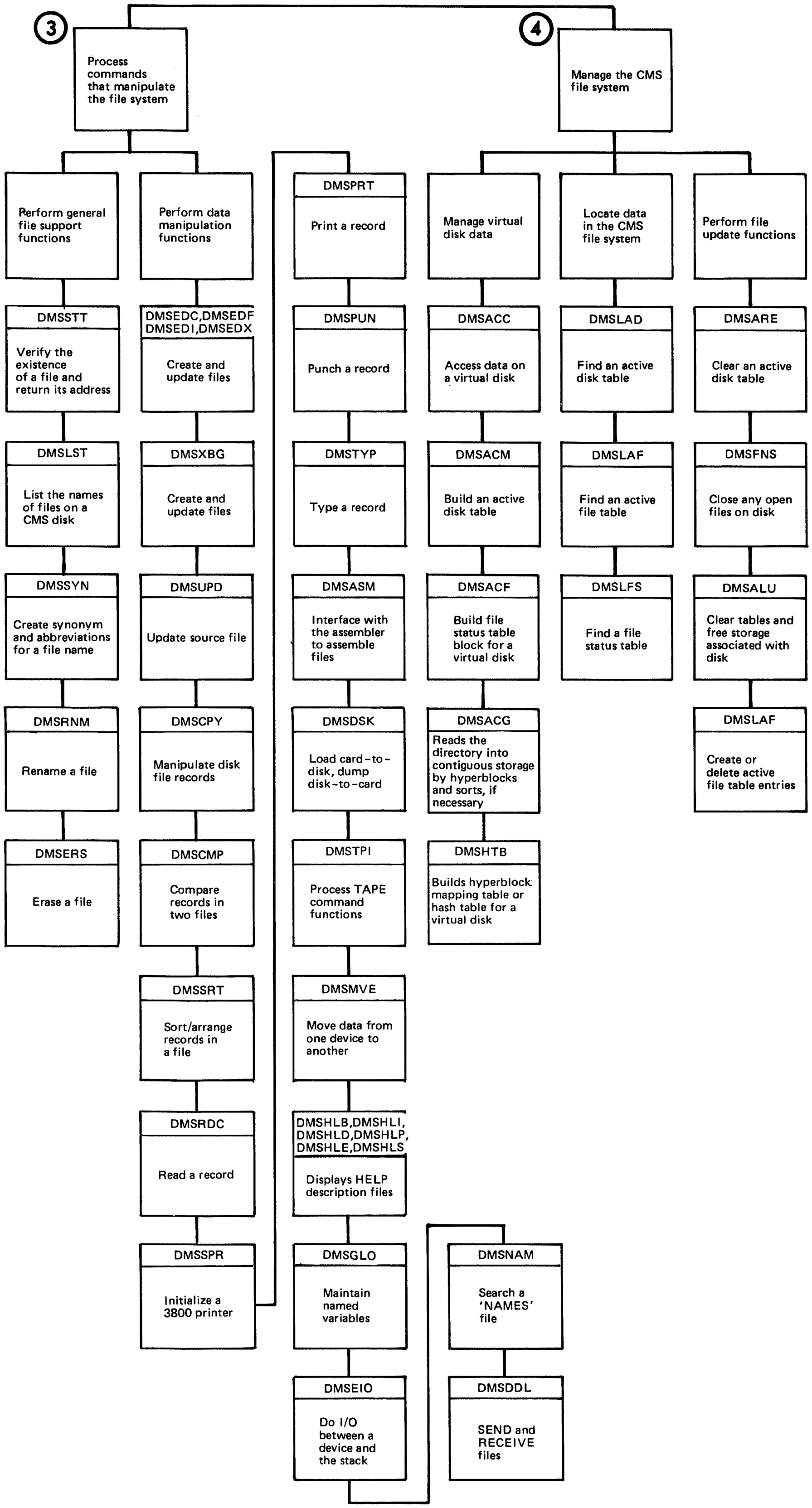

3
Process commands that manipulate the file system
4
Manage the CMS file system
Perform general file support functions
DMSSTT
Verify the existence of a file and return its address
DMSLST
List the names of files on a CMS disk
DMSSYN
Create synonym and abbreviations for a file name
DMSRNM
Rename a file
DMSERS
Erase a file
Perform data manipulation functions
DMSEDC,DMSEDF DMSEDI,DMSEDX
Create and update files
DMSXBG
Create and update files
DMSUPD
Update source file
DMSCPY
Manipulate disk file records
DMSCMP
Compare records in two files
DMSSRT
Sort/arrange records in a file
DMSRDC
Read a record
DMSSPR
Initialize a 3800 printer
DMSPRT
Print a record
DMSPUN
Punch a record
DMSTYP
Type a record
DMSASM
Interface with the assembler to assemble files
DMSDSK
Load card-to-disk, dump disk-to-card
DMSTPI
Process TAPE command functions
DMSMVE
Move data from one device to another
DMSHLB,DMSHLI, DMSHLD,DMSHLP, DMSHLE,DMSHLS
Displays HELP description files
DMSGLO
Maintain named variables
DMSEIO
Do I/O between a device and the stack
DMSNAM
Search a 'NAMES' file
DMSDDL
SEND and RECEIVE files
Manage virtual disk data
DMSACC
Access data on a virtual disk
DMSACM
Build an active disk table
DMSACF
Build file status table block for a virtual disk
DMSACG
Reads the directory into contiguous storage by hyperblocks and sorts, if necessary
DMSHTB
Builds hyperblock mapping table or hash table for a virtual disk
Locate data in the CMS file system
DMSLAD
Find an active disk table
DMSLAF
Find an active file table
DMSLFS
Find a file status table
Perform file update functions
DMSARE
Clear an active disk table
DMSFNS
Close any open files on disk
DMSALU
Clear tables and free storage associated with disk
DMSLAF
Create or delete active file table entries

Figure 8 (Part 2 of 5). Details of CMS System Functions

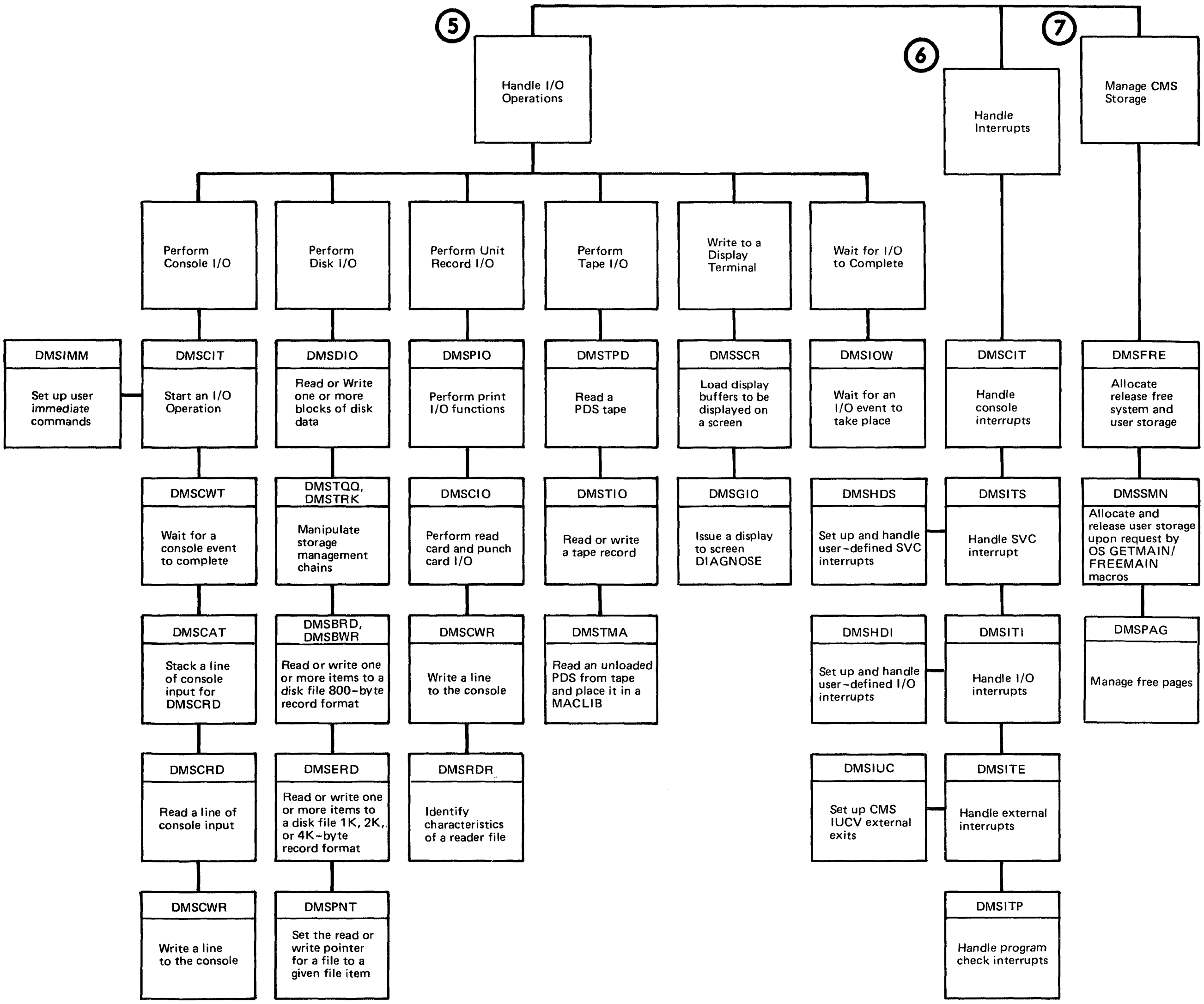

5
Handle I/O Operations
6
Handle Interrupts
7
Manage CMS Storage
Perform Console I/O
Perform Disk I/O
Perform Unit Record I/O
Perform Tape I/O
Write to a Display Terminal
Wait for I/O to Complete
DMSIMM
Set up user immediate commands
DMSCIT
Start an I/O Operation
DMSDIO
Read or Write one or more blocks of disk data
DMSPIO
Perform print I/O functions
DMSTPD
Read a PDS tape
DMSSCR
Load display buffers to be displayed on a screen
DMSIOW
Wait for an I/O event to take place
DMSCIT
Handle console interrupts
DMSFRE
Allocate release free system and user storage
DMSCWT
Wait for a console event to complete
DMSTQQ, DMSTRK
Manipulate storage management chains
DMSCIO
Perform read card and punch card I/O
DMSTIO
Read or write a tape record
DMSGIO
Issue a display to screen DIAGNOSE
DMSHDS
Set up and handle user-defined SVC interrupts
DMSITS
Handle SVC interrupt
DMSSMN
Allocate and release user storage upon request by OS GETMAIN/ FREEMAIN macros
DMSCAT
Stack a line of console input for DMSCRD
DMSBRD, DMSBWR
Read or write one or more items to a disk file 800-byte record format
DMSCWR
Write a line to the console
DMSTMA
Read an unloaded PDS from tape and place it in a MACLIB
DMSHDI
Set up and handle user-defined I/O interrupts
DMSITI
Handle I/O interrupts
DMSPAG
Manage free pages
DMSCRD
Read a line of console input
DMSERD
Read or write one or more items to a disk file 1K, 2K, or 4K-byte record format
DMSRDR
Identify characteristics of a reader file
DMSIUC
Set up CMS IUCV external exits
DMSITE
Handle external interrupts
DMSCWR
Write a line to the console
DMSPNT
Set the read or write pointer for a file to a given file item
DMSITP
Handle program check interrupts

Figure 8 (Part 3 of 5). Details of CMS System Functions

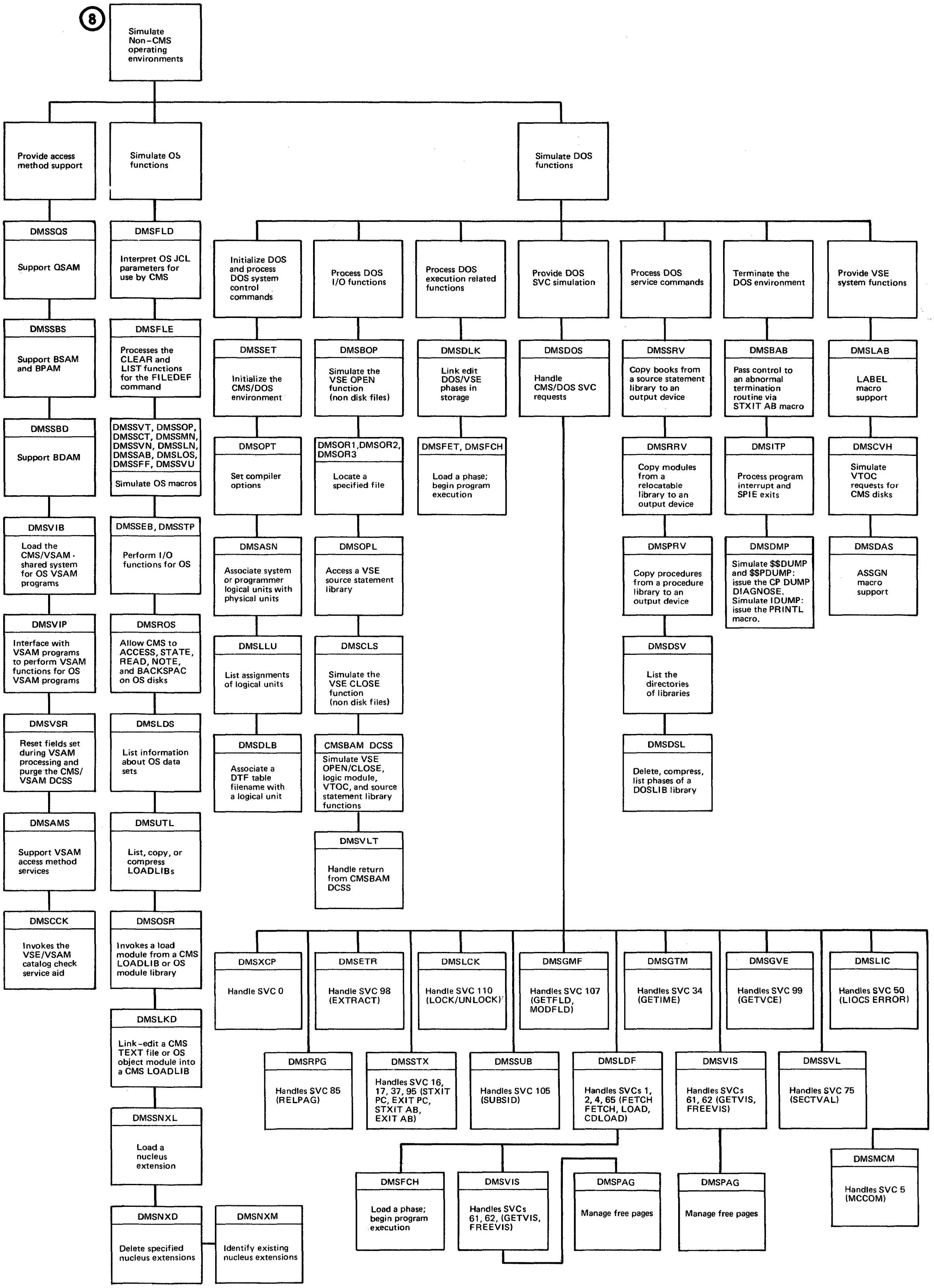

8
Simulate Non-CMS operating environments
Provide access method support
Simulate OS functions
Simulate DOS functions
DMSSQS
Support QSAM
DMSSBS
Support BSAM and BPAM
DMSSBD
Support BDAM
DMSVIB
Load the CMS/VSAM-shared system for OS VSAM programs
DMSVIP
Interface with VSAM programs to perform VSAM functions for OS VSAM programs
DMSVSR
Reset fields set during VSAM processing and purge the CMS/VSAM DCSS
DMSAMS
Support VSAM access method services
DMSCCK
Invokes the VSE/VSAM catalog check service aid
DMSFLD
Interpret OS JCL parameters for use by CMS
DMSFLE
Processes the CLEAR and LIST functions for the FILEDEF command
DMSSVT, DMSSOP, DMSSCT, DMSSMN, DMSSVN, DMSSLN, DMSSAB, DMSLOS, DMSSFF, DMSSVU
Simulate OS macros
DMSSEB, DMSSTP
Perform I/O functions for OS
DMSROS
Allow CMS to ACCESS, STATE, READ, NOTE, and BACKSPAC on OS disks
DMSLDS
List information about OS data sets
DMSUTL
List, copy, or compress LOADLIBs
DMSOSR
Invokes a load module from a CMS LOADLIB or OS module library
DMSLKD
Link-edit a CMS TEXT file or OS object module into a CMS LOADLIB
DMSSNXL
Load a nucleus extension
DMSNXD
Delete specified nucleus extensions
DMSNXM
Identify existing nucleus extensions
Initialize DOS and process DOS system control commands
Process DOS I/O functions
Process DOS execution related functions
Provide DOS SVC simulation
Process DOS service commands
Terminate the DOS environment
Provide VSE system functions
DMSSET
Initialize the CMS/DOS environment
DMSOPT
Set compiler options
DMSASN
Associate system or programmer logical units with physical units
DMSLLU
List assignments of logical units
DMSDLB
Associate a DTF table filename with a logical unit
DMSBOP
Simulate the VSE OPEN function (non disk files)
DMSOR1,DMSOR2, DMSOR3
Locate a specified file
DMSOPL
Access a VSE source statement library
DMSCLS
Simulate the VSE CLOSE function (non disk files)
CMSBAM DCSS
Simulate VSE OPEN/CLOSE, logic module, VTOC, and source statement library functions
DMSVLT
Handle return from CMSBAM DCSS
DMSDLK
Link edit DOS/VSE phases in storage
DMSFET, DMSFCH
Load a phase; begin program execution
DMSDOS
Handle CMS/DOS SVC requests
DMSSRV
Copy books from a source statement library to an output device
DMSRRV
Copy modules from a relocatable library to an output device
DMSPRV
Copy procedures from a procedure library to an output device
DMSDSV
List the directories of libraries
DMSDSL
Delete, compress, list phases of a DOSLIB library
DMSBAB
Pass control to an abnormal termination routine via STXIT AB macro
DMSITP
Process program interrupt and SPIE exits
DMSDMP
Simulate $$DUMP and $$PDUMP: issue the CP DUMP DIAGNOSE. Simulate IDUMP: issue the PRINTL macro.
DMSLAB
LABEL macro support
DMSCVH
Simulate VTOC requests for CMS disks
DMSDAS
ASSGN macro support
DMSXCP
Handle SVC 0
DMSETR
Handle SVC 98 (EXTRACT)
DMSLCK
Handle SVC 110 (LOCK/UNLOCK)
DMSGMF
Handles SVC 107 (GETFLD, MODFLD)
DMSGTM
Handles SVC 34 (GETIME)
DMSGVE
Handles SVC 99 (GETVCE)
DMSLIC
Handles SVC 50 (LIOCS ERROR)
DMSRPG
Handles SVC 85 (RELPAG)
DMSSTX
Handles SVC 16, 17, 37, 95 (STXIT PC, EXIT PC, STXIT AB, EXIT AB)
DMSSUB
Handles SVC 105 (SUBSID)
DMSLDF
Handles SVCs 1, 2, 4, 65 (FETCH FETCH, LOAD, CDLOAD)
DMSVIS
Handles SVCs 61, 62 (GETVIS, FREEVIS)
DMSSVL
Handles SVC 75 (SECTVAL)
DMSFCH
Load a phase; begin program execution
DMSVIS
Handles SVCs 61, 62, (GETVIS, FREEVIS)
DMSPAG
Manage free pages
DMSPAG
Manage free pages
DMSMCM
Handles SVC 5 (MCCOM)

Figure 8 (Part 4 of 5). Details of CMS System Functions

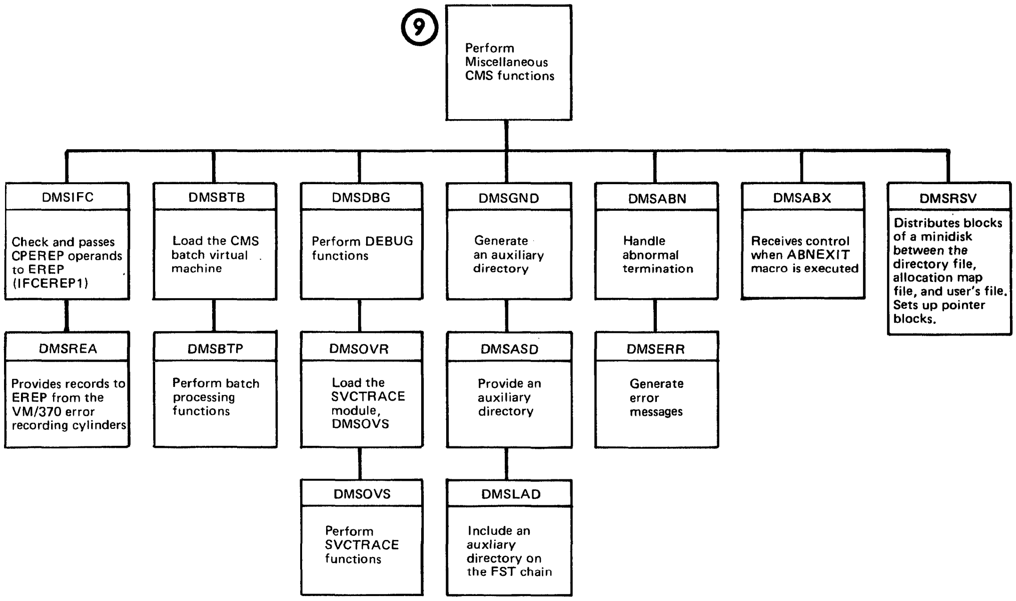

9
Perform Miscellaneous CMS functions
DMSIFC
Check and passes CPEREP operands to EREP (IFCEREP1)
DMSREA
Provides records to EREP from the VM/370 error recording cylinders
DMSBTB
Load the CMS batch virtual machine
DMSBTP
Perform batch processing functions
DMSDBG
Perform DEBUG functions
DMSOVR
Load the SVCTRACE module, DMSOVS
DMSOVS
Perform SVCTRACE functions
DMSGND
Generate an auxiliary directory
DMSASD
Provide an auxiliary directory
DMSLAD
Include an auxliary directory on the FST chain
DMSABN
Handle abnormal termination
DMSERR
Generate error messages
DMSABX
Receives control when ABNEXIT macro is executed
DMSRSV
Distributes blocks of a minidisk between the directory file, allocation map file, and user's file. Sets up pointer blocks.

Figure 8 (Part 5 of 5). Details of CMS System Functions

Chapter 6. CMS Virtual Machine Initialization

Some steps involved in initializing a CMS virtual machine are:

- Processing the IPL command for a virtual card reader
- Processing the IPL command for a disk device or a named or saved system
- Processing the first command line entered at the CMS virtual console
- Setting up the options for the virtual machine operating environment.

DMSINI and DMSINS are the two routines that are mainly responsible for the one-time initialization process in which the virtual card reader is initial program loaded (IPLed). DMSINS is called by DMSINI if the IPL is by device address. DMSINS also handles the IPL process when a named or saved system is loaded. If the IPL is by saved system name, DMSINS receives control directly from CP at the point immediately following the SAVESYS instruction.

DMSINS stacks a command to invoke the SYSPROF EXEC, if it exists, unless the user specifies the NOSPROF parameter or unless the user IPLs a non-DASD device such as a virtual reader. If it is stacked, it is invoked before any user disks are accessed. The SYSPROF EXEC contains some of the same CMS initialization function as in DMSINS, and the SYSPROF EXEC is responsible for processing the first command line entered at the CMS virtual console and accessing the user disks.

DMSINS passes control to DMSINT (the CMS command interpreter), but if the SYSPROF EXEC command is not stacked, DMSINT processes the first line entered from the console as a special case. The processing performed by this code is a part of the initialization process. DMSSET sets up the user-specified virtual machine environment features; DMSQRY allows the user to query the status of these settings.

Initialization: Loading a CMS Virtual Machine from Card Reader

When a virtual card reader is specified on the IPL command, for example 00C, initialization processing begins. Initialization refers to the process of loading from a card reader as opposed to reading a nucleus from a cylinder of a CMS minidisk or reading a named or shared system (description follows).

IPL 00C invokes DMSINI and DMSINQ. DMSINI and DMSINQ prompt the user for information used in building the CMS nucleus.

When all questions are answered, the requested nucleus is written to the DASD.

Once written on the DASD, a copy of the nucleus is read into virtual machine storage. One track at a time is read from the disk-resident nucleus into virtual storage. DMSINS is then invoked to initialize storage constants and to set up the disks and storage space required by this virtual machine.

DMSINI passes a parameter to DMSINS to indicate whether the IPL was of a DASD or non-DASD device and whether or not the user wants to save the system. If DMSINI is entered at the entry point DMSINSW, a flag will be appended to the parameters currently passed, to indicate that the IPL is of a non-DASD device, and the SYSPROF EXEC processing should be bypassed. If the user wants to save the system, this flag will also indicate this condition and will point to the saved system name.

Some of the functions that DMSINS performs include:

- Initializing storage constants and system tables

- Processing IPL command line parameters (BATCH, NOSPROF, and AUTOCR).

Initializes Storage Contents and System Tables

DMSINS
Saves the address of this virtual machine in NUCON.

DMSFRE
Allocates free storage to be used during initialization.

DMSIND
Allocates all low free nucleus storage so the system status table (S-STAT) can be built in high free storage.

Reads the S-disk ADT entry and builds the S-STAT. Reads the Y-disk ADT entry and builds the Y-STAT.

Releases the low nucleus free storage allocated above (to force SSTAT into high storage) so it can be used again.

Stores the address of S-STAT into ASSTAT and ADTFDA in NUCON. DMSINS calls DMSHTB to build hyperblock mapping tables for S- and Y-disks (if the file status tables (FSTs) span three or more pages).

Sorts the entries in the S-STAT and Y-STAT.

Processes IPL Command Line Parameters

DMSINS

Checks for parameters BATCH, INSTSEG, AUTOCR, NOSPROF, and SAVESYS.

If BATCH is specified, DMSINS sets the flag BATFLAGS.

If INSTSEG is specified, DMSINS checks for parameters YES, NO, and name. For YES or name, DMSINS calls DMSEXLSS, which accesses the Installation DCSS or the specified DCSS and loads the EXECs from it. For NO, the Installation DCSS is not accessed and the EXECs are not loaded.

If AUTOCR is specified, a local flag is set so that the subsequent console read may be bypassed and the null line input simulated. This action causes a PROFILE EXEC to be executed.

If the NOSPROF parameter is specified, a local flag is set to indicate that the SYSPROF EXEC should not be stacked, and all CMS initialization should be done by DMSINS.

If the SAVESYS parameter is specified with nothing but an associated systemname, DMSINS saves the CMS system. Any unrecognized parameters are skipped over, and parameter processing continues until the end of the line.

Issues DIAGNOSE code X'24' to obtain the device type of the console.

Issues DIAGNOSE code X'60' to get the size of the virtual machine and sets up enough storage for this virtual machine. Sets the FREELOWE pointer to NUCON.

Performs time-of-day processing and OS initialization.

DMSIND

A validity check is performed when a saved system is IPLed to ensure that the saved copy of the S-STAT or Y-STAT is current. This check is performed only for S-disks and Y-disks formatted in 512-, 1024-, 2048-, or 4096-byte CMS blocks. For 880-byte block disks, the saved copy of the S-STAT or Y-STAT is used.

A validity check consists of comparing the date that the saved directory was last updated with the date that the current disk was last updated. If the dates for the S-STAT are different, the S-STAT is built in your storage. If the dates for the Y-STAT are different, the Y-disk is accessed using the CMS ACCESS command:

```
ACCESS 19E Y/S * * Y2
```

This means that even when the S- and Y-disks are accessed in read/write mode and then released, the message DMSINS100W S-STAT and/or Y-STAT not available will result.

The DASD address of the Y-disk is whatever was specified when CMS was generated. For the standard systems it is 19E.

DMSINS

Issues HNDIUCV SET to establish CMS as an IUCV user.

Issues DIAGNOSE code X'B0' to determine if this user was automatically re-IPLed by CP and to retrieve restart information.

If not bypassing the SYSPROF EXEC, then DMSINS stacks the command to invoke it, and passes the following additional information to the SYSPROF EXEC:

CMS system id
Parameter to indicate whether S-STATs and Y-STATs are available
Indication of whether or not the 192 D disk should be accessed
Restart information from CP if this is a re-IPLed user
Parameters to indicate that SAVESYS parameter was not specified correctly.
Indication of whether IUCV initialization was completed successfully or not.

DMSCWR or SYSPROF EXEC

Writes the system id message to the console.

DMSCRD or SYSPROF EXEC

Reads the initial CMS command line from the console.

DMSSCN

Puts the initial CMS command line in PLIST format.

DMSINS or SYSPROF EXEC

Issues ACCESS 195 A to access the batch virtual machine A-disk.

DMSINS or SYSPROF EXEC

If the BATCH virtual machine is not being loaded, it determines whether there is a PROFILE EXEC or a first command line to be handled. If there is a first command, it is stacked. If there is a PROFILE EXEC, DMSINS stacks the command to invoke it and passes control to DMSINT, the CMS console manager. If the SYSPROF EXEC finds a PROFILE EXEC, it invokes the user profile directly, then exits.

Initializing a Named or Saved System

The CMS system is designed to be used as a saved, shared system. A named system is a copy of the nucleus that has been saved and named with the CP SAVESYS command. It is faster to IPL a named system than to IPL by disk address because CP maintains the named system in page format instead of CMS disk format. The initialization of a saved system is also faster because the SSTAT and YSTAT are already built.

The shared system is a variant of the saved system. In the shared system, reentrant portions of the nucleus are placed in storage pages that are available to all users of the shared system. Each user has his own copy of nonreentrant portions of the nucleus. The shared pages are protected by CP and may not be altered by any virtual machine.

During DMSINI processing, the virtual machine operator is asked if the nucleus must be written (vis message DMSINI607R). If the operator answers no, control passes directly to DMSINS to initialize the named or saved system specified by the operator in his answer to message DMSINI606R.

Modifying a 3800 Named System

The IMAGEMOD command allows an installation to modify an existing 3800 named system without the need for generating from scratch a completely new one.

The format of the IMAGEMOD command is:

```
IMAGEMOD  { GEN | ADD | REP | DEI | MAP }
              libname
              modname [ modname ] ...
              [ TERM | PRINT | DISK ]
```

For further information, refer to the *VM/SP CP for System Programming.*

Processing the IMAGEMOD Command

Module DMSIMA performs the following steps when processing the IMAGEMOD command:

1. Analyze the input PLIST for syntax. If there is an error, exit with a return code of 2 and issue the appropriate message:

   - DMSIMA001E = NO MODULE NAME SPECIFIED
   - DMSIMA003E = INVALID OPTION 'option'
   - DMSIMA014E = INVALID FUNCTION 'function'
   - DMSIMA046E = NO LIBRARY NAME SPECIFIED
   - DMSIMA047E = NO FUNCTION SPECIFIED.

2. Obtain maximum storage area (via GETMAIN macro).

3. Unless the GEN function is specified, read the named system into storage just obtained with DIAGNOSE code X'74'. Leave the first 10 pages of storage empty. This permits later expansion by 10 members.

4. Determine the type of function requested:

   - MAP
   - DEL
   - GEN
   - ADD
   - REP.

5. If the function requested is MAP, scan the named system directory and format the following information about each member:

   - Name
   - Relative displacement
   - Total size.

   Determine the option requested. If the option is TERM, PRINT, or DISK, place the formatted information on the user's terminal, virtual printer, or in the CMS file named 'libname MAP A5', respectively.

6. If the function requested is DEL, delete the member from the directory and the data area of the named system. Compress the named system by moving up the remaining members to take up the space vacated by the deletion. If the member is not found, issue message DMSIMA013E.

7. If the function requested is GEN, construct a skeleton named system in virtual storage. This skeleton system has no members initially. Then proceed as if the function were ADD.

8. If the function requested is ADD, load the member into the CMS transient area. If a load error occurs, issue DMSIMA346E and exit with return code of 6. Add the new member entry to the end of the named system directory. If virtual capacity is exceeded by this addition, issue DMSIMA109E and exit with return code of 2. During this process, the directory is moved back in storage one page to prevent new data from overlaying existing data. Move the new member data to the end of the named system residing in user virtual storage. Modify the directory entries after this move takes place. If the member already exists, issue message DMSIMA751E and exit with return code of 4.

9. If the function requested is REP, concatenate the DEL and ADD functions. In other words, perform the DEL function and the the ADD function for the specified member.

10. Scan the input command line for more members to be processed. If there are no more members or if the number of members has reached the maximum (10), write the changed named system back to disk via DIAGNOSE code X'74' (unless this was a MAP function request) and then exit. Otherwise, process the next member according to the function requested.

Handling the First Command Line Passed to CMS

DMSINT, the CMS console manager, contains the code to handle commands stacked by module DMSINS during initialization processing. DMSINT checks for the presence of a stacked command line, and if there is one to process, DMSINT processes it just as it would a command entered during a terminal session. That is, DMSINT calls the WAITREAD subroutine and issues an SVC 202 to execute the command. When first command processing completes, DMSINT receives control to handle commands entered at the console for the duration of the session.

Setting the Virtual Machine Environment Options

DMSSET sets up the virtual machine environment options. This module is structured and relatively easy to follow, except for some sections of DMSSET.

DMSSET: Set DOS ON (VSAM) Processing

DMSSET
(label DOS) If a disk mode is specified on the command line, ensures that it is valid.

DMSLAD
If the disk mode specified is valid, locates and returns the address of the disk.

DMSSET
Issues DIAGNOSE code X'64' FINDSYS to locate the CMSDOS or CMSBAM segments. If the segment is not already loaded, issues DIAGNOSE code X'64' LOADSYS to load it.

DMSSET
Sets up the $$B-transient area for use by VSE routines.

DMSSET
Sets up the LOCK/UNLOCK resource table.

DMSSET
If SET DOS OFF has been specified, issues the DIAGNOSE code X'64' PURGESY function for the CMSDOS and CMSBAM segments and, if VSAM has been loaded, for the CMSVSAM segment.

Querying CMS Environment Options

The QUERY command, which displays CMS environment options, is handled by eight modules. DMSQRY is the main module. The first time QUERY is invoked, DMSQRY established QUERY as a nucleus extension. DMSQRY acquires a work area and uses DMSQRZ to initialize it.

If the option queried is a CMS option and if the command has the correct syntax, DMSQRY passes control to the module that handles that option: DMSQRS, DMSQRT, DMSQRU, DMSQRV, DMSQRW, or DMSQRX. The module called performs the requested QUERY function, then returns control to the original caller.

Chapter 7. Processing and Executing CMS Files

As shown in Part 1 of Figure 8 on page 59 five general topics form the category "Process and Execute CMS Files". Two of these topics are discussed in this section: "Maintaining an Interactive Console Environment" and "Loading and Executing Text Files" on page 94.

Maintaining an Interactive Console Environment

Two levels of information are discussed in the following section. The first level is a general discussion of how CMS maintains an interactive console environment. The second level is a more detailed discussion of the methods of operation mainly responsible for this function.

There are two major functions concerned with maintaining an interactive terminal environment for CMS: console management and command processing. The CMS module that manages the virtual machine console is DMSINT. The module responsible for command processing is DMSITS. Many CMS modules are called in support of these two functions, but the modules in the following list are primarily responsible for supporting the functions:

DMSCRD
Reads a line from the console.

DMSCWR
Writes a line to the console.

DMSSCN
Converts a command line to PLIST format.

DMSPKT
Translates command names.

DMSINA
Converts abbreviated commands to their full names.

DMSCPF
Passes a command line to CP for execution.

Maintaining an Interactive Command/Response Session

Three main lines of control maintain the continuity for an interactive CMS session: (1) handling of commands passed to DMSINT by the initialization module, DMSINS (2) handling of commands entered at the console during a session, and (3) handling of commands entered as subset commands. The following lists show the main logic paths for the first two functions.

Execute Commands Passed via DMSINS

DMSINT
On entry from DMSINS, processes any commands passed via the console read put on the user's console by that routine. That is, processes any commands the user stacks on the line as the first read that DMSINT processes. In handling the first read, if that read is null, control passes to the main loop of the program, which is described in the following section.

DMSINM
Retrieves the current time.

DMSCRD
Branches to the waitread subroutine to read a command line at the console.

DMSSCN
Waitread then calls DMSSCN to convert the line just read into PLIST format. Once converted to PLIST format, an SVC 202 is issued (at label INIT1A) to execute the function. This cycle is repeated until all stacked commands are executed.

DMSFNS
When command execution completes, calls DMSFNS (at label UPDAT) to close any files that may have remained open during the command processing.

DMSVSR
Ensures that any fields set by VSAM processing are reset for CMS. Also ensures that the VSAM discontiguous shared segment is purged.

DMSINT
Sets up an appropriate status message (CMS, CMS SUBSET, CMS/DOS, etc.).

DMSCWR
Writes the status message to the console.

Handle Commands Entered During a CMS Terminal Session

DMSINT
Branches (from label INLOOP2) to the waitread subroutine to read a line entered at the console.

DMSCRD
Reads a line entered at the console (subroutine waitread).

DMSSCN
Converts the command line to PLIST format (subroutine waitread).

DMSINT
Determines whether the command line is a null line or a comment.

DMSLFS
If the command line is neither a command line nor a comment, determines whether the command is an EXEC file.

DMSINA (ABBREV)
Determines whether the command is an abbreviation, and if it is, returns its full name.

DMSITS
Passes the command line to DMSITS via an SVC 202. DMSITS is the CMS SVC handler. For a detailed description of the SVC handler, see "Method of Operation for DMSITS - CMS SVC Handling Routine" on page 77.

DMSCPF
If the command could not be executed by the SVC handler, passes the command to CP to see if CP can execute it.

DMSFNS
On return from processing the command line (label UPDAT), closes any files that may have been opened during processing.

DMSSMN
Resets any flags or fields that may have been set during OS processing.

DMSVSR
Ensures that any fields set for VSAM processing are reset for CMS. Also ensures that the VSAM discontiguous shared segment is purged.

DMSINT
When the command line has been successfully executed, builds a CMS ready message for the user (label PRNREADY).

DMSCWR
Writes the ready message to the console.

DMSINT
Returns control to DMSINT at label INLOOP2 to continue monitoring the CMS terminal session.

Method of Operation for DMSINT - Console Manager

DMSINT, the console manager, maintains the continuity of operation of the CMS command environment. The main control loop of DMSINT is initiated by a call to DMSCRD to get the next command. When the command is entered, DMSINT calls DMSINM to initialize the CPU time for the new command and then puts it in both a standard tokenized and an extended parameter list form by calling the scan function program DMSSCN. After calling DMSSCN, DMSINT checks to see if an EXEC filetype exists with a filename of the type-in command. (For example, if ABC was typed in, it checks to see if ABC EXEC exists.) If the EXEC file does exist, DMSINT adjusts register 1 to point to the same command set up by DMSSCN, but preceded by CL8'EXEC'. Then DMSINT issues an SVC 202 to call the corresponding EXEC procedure ('ABC EXEC' in the example).

If no such EXEC file exists for the first word typed in, DMSINT checks for a translation via DMSPKT. If no translation or synonym is found, DMSINT makes a further check using the CMS abbreviation-check routine, DMSINA. If the translation or synonym is found, substitute it for the typed-in word. If, for example, the first word typed in had been 'E', DMSINT looks up 'E' via DMSPKT. If not found, then DMSINT looks up 'E' via DMSINA. If an equivalent is found for 'E', DMSINT looks for an EXEC file with the name of the equivalent word (for example, EDIT EXEC). If such a file is found, DMSINT adjusts register 1 as described above to call the EXEC and substitutes the equivalent word, EDIT, for the first word typed in. Thus, if 'E' is a valid abbreviation for 'EDIT' and you have an EXEC file called EDIT EXEC, EDIT EXEC is invoked when you type in 'E' from the terminal.

If no EXEC file is found either for the entered command name or for any equivalent found by DMSINA or DMSPKT, DMSINT leaves the terminal command as processed by DMSSCN and then issues an SVC 202 to pass control to DMSITS. DMSITS then passes control to the appropriate command program. When the command terminates execution, or if DMSITS cannot execute it, the return code is passed in register 15.

A zero return code indicates successful completion of the command. A positive return code indicates that the command was completed, but with an apparent error. A negative code returned by DMSITS indicates that the typed in command could not be found or executed at all.

In the last case, DMSINT assumes that the command is a CP command and issues a DIAGNOSE instruction to pass the command line to the CP environment. If the command is not a CP command, DMSINT calls DMSCWR to type a message indicating that the command is unknown and the main control loop of DMSINT is entered at the beginning.

If the return code from DMSITS is positive or zero, DMSINT saves the return code briefly and calls module DMSAUD to update the master file directory (MFD) on the appropriate user's disk for the 800-byte records on disk, or to update the file directory and the allocation map, or the appropriate user's disk for the 512-, 1K-, 2K-, or 4K-byte records on disk. DMSINT also frees the TXTLIB chain and releases pages of storage if required.

After updating the file directory, DMSINT checks the return code that was passed back. If the code is zero, DMSINT types a ready message and the processor time used by the given command. Control is passed to the beginning of the main control loop of DMSINT. If the return code is positive, an error message is typed, along with the processor time used. The command causes the typing of an error message with the format: DMSxxxnnnt 'text' where DMSxxx is the module name, nnn is the message identification number, t is the message type, and 'text' is the message explaining the error. Control is then passed to the beginning of the main control loop.

Method of Operation for DMSITS - CMS SVC Handling Routine

DMSITS (INTSVC) is the CMS system SVC handling routine. The general operation of DMSITS is as follows:

1. The SVC new PSW (low-storage location X'60') contains, in the address field, the address of DMSITS1. Thus, the DMSITS routine is entered whenever a supervisor call is executed.

2. DMSITS allocates a system save area and a user save area. The user save area is a register save area used by the routine, which is invoked later as a result of the SVC call.

3. The called routine is invoked (via a LPSW or BALR).

4. Upon return from the called routine, the save areas are released.

5. Control is returned to the caller (the routine that originally made the SVC call).

Types of SVCs and Linkage Conventions

The types of SVC calls recognized by DMSITS, and the linkage conventions for each, are as follows:

SVC 201: When a called routine returns control to DMSITS, the user storage key may be in the PSW. Because the called routine may also have turned on the problem bit in the PSW, the most convenient way for DMSITS to restore the system PSW is to cause another interruption, rather than to attempt the privileged Load PSW instruction. DMSITS does this by issuing SVC 201, which causes a recursive entry into DMSITS. DMSITS determines if the interruption was caused by SVC 201, and if so, determines if the SVC 201 was from within DMSITS. If both conditions are met, control returns to the instruction following the SVC 201 with a PSW that has the problem bit off and the system key restored.

SVC 202: SVC 202 is the most commonly used SVC in the CMS system. It is used for calling nucleus-resident routines, nucleus extensions, and routines written as commands (for example, disk resident modules).

A typical coding sequence for an SVC 202 call is the following:

```
LA   R1,PLIST
SVC 202
DC   AL4(ERRADD)
```

The "DC AL4(address)" following the SVC 202 is optional and may be omitted if the programmer does not expect any errors to occur in the routine or command being called. If the DC statement is included, an error return is made to the address specified in the DC, unless the address is equal to 1. If the address is 1, return is made to the next instruction after the "DC AL4(1)" instruction. DMSITS determines whether this DC was inserted by examining the byte following the SVC call inline. If the byte is nonzero, the statement following the SVC 202 is an instruction. If the byte is zero, then the statement is a "DC AL4(address)" or "DC AL4(1)".

If you want to ignore errors, use the following sequence:

```
LA   R1,PLIST
SVC 202
DC   AL4(1)
```

Whenever SVC 202 is called, the contents of Register 0 and Register 1 are passed intact to the called routine. Register 1 must point to an eight-character string, which may be the start of a tokenized PLIST. This character string must contain the symbolic name of the routine or command being called. The called routine decides whether to use the tokenized PLIST or the extended PLIST (one of two forms) by examining the high-order byte of R1. (Both forms of the extended PLIST are discussed below.) The SVC handler only examines the name and high-order byte of Register 1.

Note: Although an extended PLIST is provided, the called routine may not be set up to use it.

Value	Meaning	Extended PLIST Pointer in Register 0?
X'00'	The call did not originate from an EXEC file or a command typed at the terminal. (The SVC handler translates the value X'04' to X'00' before entering the called program.)	No
X'01'	Either, the call is from an EXEC 2 EXEC or the System Product Interpreter when "ADDRESS COMMAND" is specified, or the call is an IBM Cooperative Processing for VM/SP call (see SENDREQ in the *VM/SP Programmer's Guide to the Server-Requester Programming Interface for VM/SP,* SC24-5291). You can tell by checking the form of the extended PLIST, see "Extended PLIST" on page 80. (The SVC handler translates the value X'03' to X'01' before entering the called program.)	Yes
X'02'	See "Dynamic Linkage/SUBCOM" in this manual.	Yes
X'05'	Used by the System Product Interpreter for external function calls.	Yes
X'06'	The command was invoked as an immediate command. This setting should never occur with SVC 202.	Yes
X'0B'	The command was called as a result of its name being typed at the terminal, by the CMDCALL command to invoke the command from EXEC 2, or from a System Product Interpreter program when "ADDRESS CMS" is specified.	Yes
X'0C'	The call is the result of a command invoked from a CMS EXEC file with "&CONTROL" set to something other than "NOMSG" or "MSG".	No
X'0D'	The call is the result of a command invoked from a CMS EXEC file with "&CONTROL MSG" in effect (indicates that messages are to be displayed at the terminal).	No
X'0E'	The call is the result of a command invoked from a CMS EXEC file with "&CONTROL NOMSG" in effect.	No
X'FE'	This is an end-of-command call from DMSINT (CMS console command handler). See the NUCEXT function in the *VM/SP CMS Macros and Functions Reference* for details.	No
X'FF'	This is a service call from DMSABN (abend) or from NUCXDROP. See the NUCEXT function in the *VM/SP CMS Macros and Functions Reference* for details.	No

Figure 9. SVC 202 High-Order Byte Values of Register 1

Tokenized PLIST: For a tokenized parameter list, the symbolic name of the function being called (8 character string, padded with blank characters on the right if needed) is followed by extra arguments depending on the actual routine or command called. These arguments must be "tokenized." Every parenthesis is considered an individual argument, and each argument may have a maximum length of eight characters.

Extended PLIST: For an extended parameter list (EPLIST), no restriction is put on the structure of the argument list passed to the called routine or command. The first non-blank character, left parenthesis, or right parenthesis following the command is treated as a delimiter. This delimiter determines where the pointer to the start of the argument is.

An extended PLIST has two forms, as illustrated below.

In the first form, R0 points to the following parameter list:

```
(a)  DC A(COMVERB)
(b)  DC A(BEGARGS)
(c)  DC A(ENDARGS)
(d)  DC A(0)
```

where the first three addresses are defined by:

```
COMVERB EQU *
        DC C'cmdname'     name of command
BEGARGS EQU *
        DC C'        '    argument list
ENDARGS EQU *
```

and where:

(a) is the beginning address of the command
(b) is the beginning address of the argument list.
(c) is the address of the byte immediately following the end of the argument list.
(d) may be used to pass any additional information required by individual called programs. If this word is not used to pass additional information, it should be zero so that programs receiving optional information via this word may detect that none is provided in this call.

Notes:

1. *These four words can be moved to some location convenient for the command resolution routines or convenient for some other program executed between the caller's SVC 202 and entry to the program that the parameter list is intended. For this reason, the called program may not assume additional words following word 4, or the called program may not assume that the storage address of these 4 words bears any relationship to other data addresses.*

2. *For function calls in the System Product Interpreter, two additional words are available. See the VM/SP System Product Interpreter*

Reference for more information on function calls and the two additional words.

The second form of an extended PLIST is used by IBM Cooperative Processing for VM/SP (see SENDREQ in the *VM/SP Programmer's Guide to the Server-Requester Programming Interface for VM/SP,* SC24-5291). In the second form, R0 points to the following parameter list:

```
(a) DS A(commandname)
(b) DS F                    (reserved)
(c) DS F                    (reserved)
(d) DS A(CPRB)
```

where:

(a) is the address of the name of the program being called
(b) is unused
(c) is unused
(d) is the address of the cooperative processing request block (CPRB).

If your routine is being called by another routine, you can verify that your routine is being called using the second form of an extended PLIST. Check the contents of A(CPRB) + 4. At this address should contain the characters CPRB.

If you want to call another routine using the second form of an extended PLIST, see SENDREQ in the *VM/SP Programmer's Guide to the Server-Requester Programming Interface for VM/SP,* SC24-5291.

Why Use the Second Form of an Extended PLIST?: The second form provides an architected way for a routine to:

- Pass up to 64K-1 (65,535) bytes of arbitrary data and 32K-5 (32,763) bytes of parameters to another routine

- Receive up to 64K-1 (65,535) bytes of arbitrary data and 32K-5 (32,763) bytes of parameters from another routine.

SVC 203: SVC 203 is called by CMS macros to perform various internal system functions. It defines SVC calls when no parameter list is provided. For example, DMSFREE parameters are passed in registers 0 and 1.

A typical sequence for an SVC 203 call is:

```
SVC   203
DC    H'code'
```

The halfword decimal code following the SVC 203 indicates the specific routine being called. DMSITS examines this halfword code taking the absolute value of the code using an LPR instruction. The first byte of the result is ignored, and the second byte of the resulting halfword is used as an index into a branch table. The address of the correct routine is loaded, and the control is transferred to it.

It is possible for the address in the SVC 203 index table to be zero. In this case, the index entry contains an 8-byte routine or command name, which is processed in the same way as the 8-byte name passed in the parameter list to a SVC 202.

The sign of the halfword code indicates whether the programmer expects an error return. If an error return is expected, the code is negative. If the code is positive, no error return is made. The sign of the halfword code has no effect on determining the routine called since DMSITS takes the absolute value of the code to determine the called routine.

Since only the second byte of the absolute value of the code is examined by DMSITS, seven bits (bits 1-7) are available as flags or for other uses. For example, DMSFREE uses these seven bits to indicate such things as conditional requests and variable requests. Therefore, DMSITS considers the codes H'3' and H'259' to be identical and handles them the same as H'-3' and H'-259', except for error returns.

When an SVC 203 is invoked, DMSITS stores the halfword code into the NUCON location CODE203 so the called routine can examine the seven bits made available to it.

All calls made by SVC 203 should be made by macros with the macro expansion computing and specifying the correct halfword code.

User-Handled SVCS: The programmer may use the HNDSVC macro to specify the address of a routine that processes any SVC call for SVC numbers 0 through 200 and 206 through 255. If the HNDSVC macro is used, the linkage conventions are as required by the user specified SVC-handling routine. You cannot specify a normal or error return from a user-handled SVC routine.

OS Macro Simulation SVC Calls: CMS supports selected SVC calls generated by OS macros, by simulating the effect of these macro calls.

The proper linkages are set up by the OS macro generations. DMSITS does not recognize any way to specify a normal or error return from an OS macro simulation SVC call.

VSE SVC Calls: All SVC functions supported for CMS/DOS are handled by the CMS module DMSDOS. DMSDOS receives control from DMSITS (the CMS SVC handler) when that routine intercepts a VSE SVC code and finds that the DOSSVC flag in DOSFLAGS is set in NUCON.

DMSDOS acquires the specified SVC code from the OLDPSW field of the current SVC save area. Using this code, DMSDOS computes the address of the routine where the SVC is to be handled.

Many CMS/DOS routines (including DMSDOS) are contained in a discontiguous shared segment (DCSS). Most SVC codes are executed within DMSDOS, but some are in separate modules external to DMSDOS. If the SVC code requested is external to DMSDOS, its address is computed using a table called DCSSTAB. If the code requested is executed within

DMSDOS, the table SVCTAB is used to compute the address of the code to handle the SVC.

DOS SVC calls are discussed in more detail in "Simulating a VSE Environment Under CMS" on page 201.

Invalid SVC Calls: There are several types of invalid SVC calls recognized by DMSITS:

- Invalid SVC number. If the SVC number does not fit into any of the classes described above, it is not handled by DMSITS. An error message is displayed at the terminal, and control is returned directly to the caller.
- Invalid routine name in SVC 202 parameter list. If the routine named in the SVC 202 parameter list is invalid or cannot be found, DMSITS handles the situation in the same way it handles an error return from a legitimate SVC routine. The error code is -3.
- Invalid SVC 203 code. If an invalid code follows SVC 203 inline, an error message is displayed and the ABEND routine is called to terminate execution.

Search Hierarchy for SVC 202

SVC 202 Entered from a Program: When a program issues SVC 202 and passes a routine or command name in the parameter list, DMSITS searches for the specified routine or command. (In the case of SVC 203 with a zero in the table entry for the specified index, the same logic must be applied.)

As soon as the routine or command name is found, the search stops and the routine or command is executed. The search order is as follows:

1. DMSITS determines if the specified name is known dynamically to CMS through the SUBCOM function. This step is executed only if the high-order byte of R1 contains X'02'.
2. DMSITS searches for a nucleus extension routine with the specified name.

   Note: This step is skipped if the high-order byte of register 1 contains X'03' or X'04'. X'03' indicates that an extended PLIST is provided. X'04' indicates that a tokenized PLIST is provided. X'03' and X'04' are translated to X'01' and X'00', respectively, by the SVC interrupt handler before the called program is entered.
3. DMSITS searches for a routine with the specified name in the transient area.
4. DMSITS searches for a nucleus-resident command with the specified name.
5. DMSITS searches currently accessed disks for a file with the specified name and a filetype MODULE. CMS uses the standard search order (A

through Z). If this search is successful, the specified module is loaded (via the LOADMOD command) and control is passed to the storage location now occupied by the command. The table of active (open) disk files is searched first. An open file may be used ahead of a file that resides on a disk earlier in the search order.

6. DMSITS calls

   a. DMSPKT to search the translation tables for the specified name. If found, DMSITS searches for a routine with the valid translation by repeating steps 2 through 5.

      Note: This step is skipped if this SVC call is not from DMSINT or DMSCSF.

   b. DMSINA to search the synonym tables for the specified name. If found, DMSITS searches for a routine with the valid synonym by repeating steps 2 through 5.

If all searches fail, then an error code of -3 is issued.

Commands Entered from the Terminal: When a command is entered from the terminal, DMSINT processes the command line and calls the scan routine to convert it into a parameter list consisting of 8-byte entries.

As soon as the command name is found, the search stops and the command is executed. The search order is as follows:

1. Search for an EXEC with the specified command name:[2]

   a. DMSINT searches for an EXEC in storage. If an EXEC with this name is found, DMSINT determines whether the EXEC has a USER, SYSTEM, or SHARED attribute. If the EXEC has the USER or SYSTEM attribute, it is executed.

      If the EXEC has the SHARED attribute, the INSTSEG setting is checked. When INSTSEG is ON, all accessed disks are searched and the access mode of the Installation Discontiguous Shared Segment (DCSS) is compared to the mode of an EXEC with that name that resides on disk. If the access mode of the DCSS is equal to or higher than the disk mode, the EXEC is executed. Otherwise, the EXEC on disk is executed.

   b. DMSINT searches accessed disks for a file with the specified name and filetype EXEC. The table of active (open) disk files is searched first. An open file may be used ahead of a file that resides on a disk earlier in the search order.

2. DMSINT calls

   a. DMSPKT to search the translation tables for the specified name. If found, DMSINT searches for a routine with the valid translation by repeating step 1.

b. DMSINA to search the synonym tables for the specified name. If found, DMSINT searches for a routine with the valid synonym by repeating step 1.

3. DMSINT executes SVC 202, passing the scanned tokenized parameter list, with the command name in the first eight bytes of the PLIST pointed to by register 1 and the extended PLIST address in register 0. DMSITS performs the search for SVC 202 as described above in "SVC 202 Entered from a Program" on page 83.

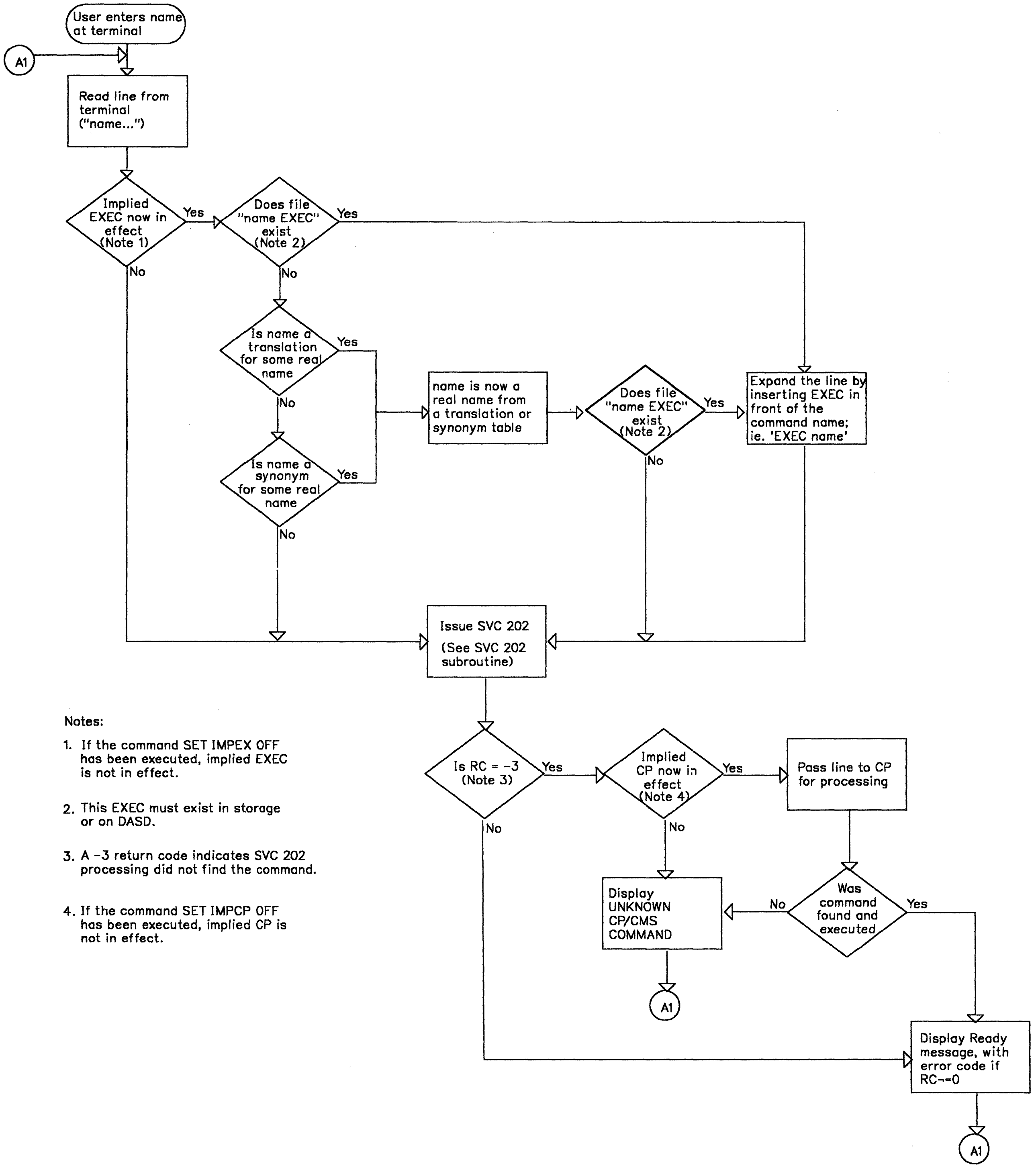
4. DMSINT searches for a CP command with the specified name, using the CP DIAGNOSE function.[3]

5. If all of these searches fail, DMSINT displays the error message `Unknown CP/CMS Command.`

See Figure 10 on page 86 for a description of this search for a command name.

[2] If implied EXEC is not in effect (SET IMPEX OFF), skip steps 1 and 2.

[3] If implied CP is not in effect (SET IMPCP OFF), skip step 4.


User enters name at terminal
A1
Read line from terminal ("name...")
Implied EXEC now in effect (Note 1)
Yes
No
Does file "name EXEC" exist (Note 2)
Yes
No
Is name a translation for some real name
Yes
No
Is name a synonym for some real name
Yes
No
name is now a real name from a translation or synonym table
Does file "name EXEC" exist (Note 2)
Yes
No
Expand the line by inserting EXEC in front of the command name; ie. 'EXEC name'
Issue SVC 202
(See SVC 202 subroutine)
Is RC = -3 (Note 3)
Yes
No
Implied CP now in effect (Note 4)
Yes
No
Pass line to CP for processing
Display UNKNOWN CP/CMS COMMAND
A1
Was command found and executed
No
Yes
Display Ready message, with error code if RC¬=0
A1

Notes:

1. If the command SET IMPEX OFF has been executed, implied EXEC is not in effect.

2. This EXEC must exist in storage or on DASD.

3. A -3 return code indicates SVC 202 processing did not find the command.

4. If the command SET IMPCP OFF has been executed, implied CP is not in effect.

Figure 10. CMS Command Processing

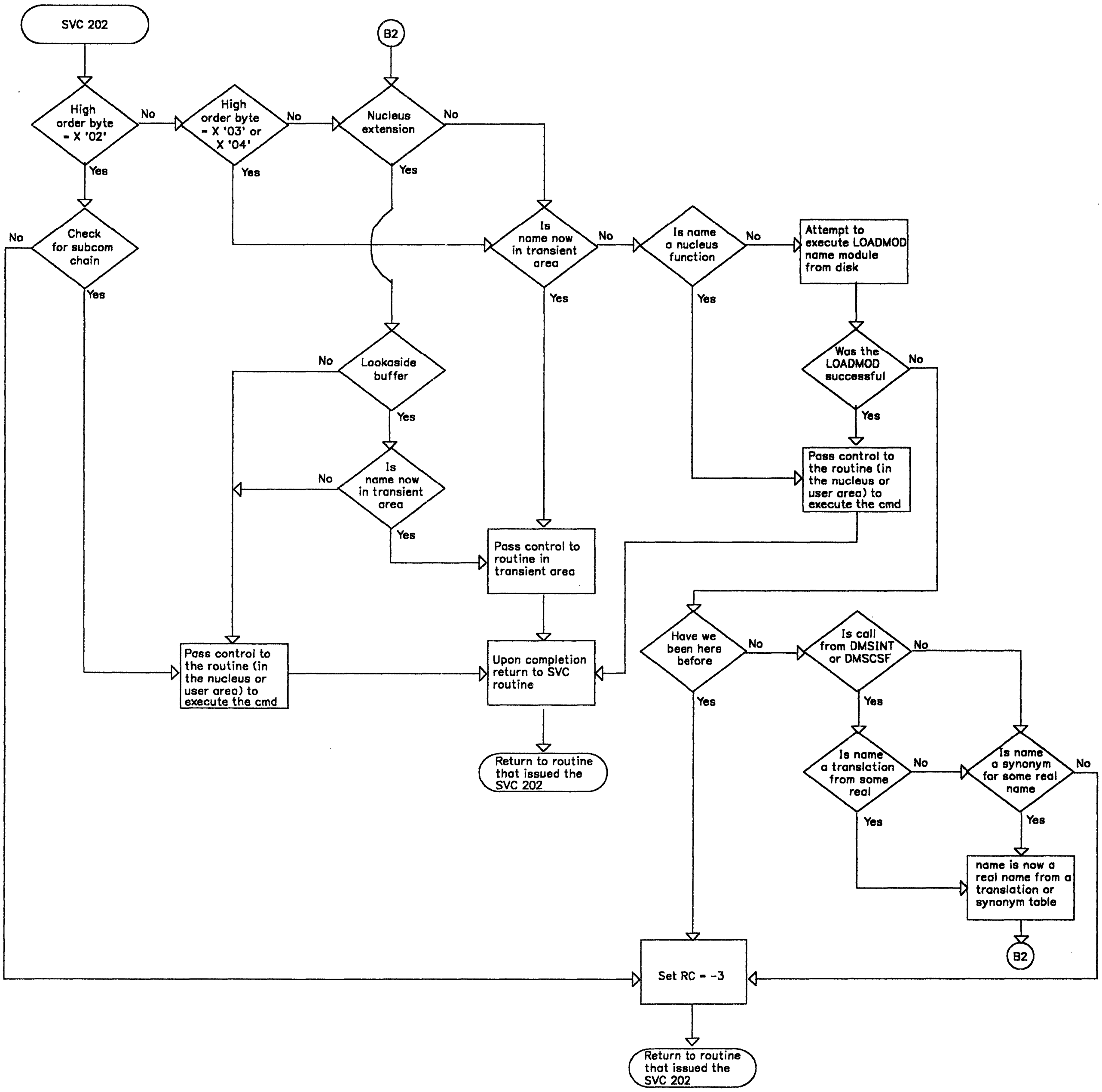

SVC 202
B2
High order byte = X '02'
No
Yes
High order byte = X '03' or X '04'
No
Yes
Nucleus extension
No
Yes
Check for subcom chain
No
Yes
Is name now in transient area
No
Yes
Is name a nucleus function
No
Yes
Attempt to execute LOADMOD name module from disk
Was the LOADMOD successful
No
Yes
Lookaside buffer
No
Yes
Is name now in transient area
No
Yes
Pass control to the routine (in the nucleus or user area) to execute the cmd
Pass control to routine in transient area
Pass control to the routine (in the nucleus or user area) to execute the cmd
Upon completion return to SVC routine
Return to routine that issued the SVC 202
Have we been here before
No
Yes
Is call from DMSINT or DMSCSF
No
Yes
Is name a translation from some real
No
Yes
Is name a synonym for some real name
No
Yes
name is now a real name from a translation or synonym table
B2
Set RC = -3
Return to routine that issued the SVC 202

Figure 11. SVC 202 Processing

User and Transient Program Areas

Two areas hold programs that are loaded by LOADMOD from the disk. These areas are called the user program area and the transient program area. (See Figure 3 on page 20 for a description of CMS storage usage.) A summary of CMS modules and their attributes, including whether they reside in the user program area or the transient area, is contained in the *VM/SP CMS Command Reference.*

The user program area starts at location X'20000' and extends upward to the loader tables. However, the high-address end of that area can be allocated as free storage by DMSFREE. Generally, all user programs and certain system commands, such EDIT and COPYFILE, are executed in the user program area. Because only one program can be executing in the user program area at one time, it is impossible (without unpredictable results) for one program executing in the user program area to invoke, by means of SVC 202, a module that will also be executed the user program area.

The transient program area is two pages, running from location X'E000' to location X'FFFF'. It provides an area for system commands that may also be invoked from the user program area by means of an SVC 202 call. When a transient module is called by an SVC, it is normally executed with the PSW system mask disabled for I/O and external interrupts.

A program executing in the transient program area may not invoke another program intended to execute in the transient program area. Thus, for example, a program executing in the transient program area may not invoke the TYPE command.

There is one further functional difference between the use of the two program areas. DMSITS starts a program in the user program area so that it is enabled for all interruptions. It starts a program in the transient program area so that it is disabled for all interruptions. Thus, the individual program may have to use the SSM (Set System Mask) instruction to change the current status of its system mask.

Called Routine Start-Up Table

Figure 12 shows how registers are set up when the called routine is entered.

Type	Registers 0 - 1	Register 2	Registers 3 - 11	Register 12	Register 13	Register 14	Register 15
SVC 202	Same as caller	See note 1	Not defined	Address of called routine	Address of user save area	Return address to DMSITS	Address of called routine
SVC 203	Same as caller	Not defined	Not defined	Address of called routine	See note 2	Return address to DMSITS	Address of called routine
Other	Same as caller	Same as caller	Same as caller	Address of called routine	Address of user save area	Return address to DMSITS	Same as caller

Figure 12. Register Contents When Called Routine Starts

Notes:

1. *If a nucleus extension or subcommand processor, register 2 has address of SCBLOCK.*

2. *Depends on the function being invoked.*

Figure 13 shows how the PSW fields are set up when the called routine is entered.

Called Type	System Mask	Storage Key	Problem Bit
SVC 202 or 203 -- Nucleus Resident	Disabled	System	Off
SVC 202 -- Nucleus Extension Module	See note 1	See note 1	Off
SVC 202 or 203 -- Transient Area Module	Disabled	See note 2	Off
SVC 202 or 203 -- User Area Module	Enabled	See note 2	Off
User-handled	Enabled	User	Off
OS-VSE -- Nucleus resident	Disabled	System	Off
OS-VSE -- Transient area module	Disabled	System	Off

Figure 13. PSW Fields When Called Routine Starts

Notes:

1. *User defined by using the NUCEXT function.*

2. *User defined by using the CMS GENMOD command or the CMS SET PROTECT command.*

Returning to the Caller

When the called routine is finished processing, it returns control to DMSITS. Then DMSITS returns control to the calling routine.

Return Location: The return is accomplished by loading the original SVC old PSW (that was saved at the time DMSITS was first entered), after possibly modifying the address field. The address field modification depends upon the type of SVC call and on whether the called routine indicated an error return address.

For SVC 202 and 203, the called routine places a zero in Register 15 indicating a normal return places a nonzero in Register 25 indicating an error return. lacing a zero in register 15 and an error return by placing a nonzero in register 15. If the called routine indicates a normal return, DMSITS makes a normal return to the calling routine. If the called routine indicates an error return, DMSITS passes the error return to the calling routine, if one was specified. If no error return address was specified, DMSITS abnormally terminates.

For SVC 202 not followed by "DC AL4(address)" or "DC AL4(1)", a normal return is made to the instruction following the SVC instruction and an error return causes an abend. For an SVC 202 followed by "DC AL4(address)", a normal return is made to the instruction following the DC and an error return is made to the address specified in the DC, unless the address is equal to 1. If the address is 1, return is made to the next instruction after the "DC AL4(1)" instruction. In either case, register 15 contains the return code passed by the called routine.

For SVC 203 with a positive halfword code, a normal return is made to the instruction following the halfword code and an error return causes an abend. For SVC 203 with a negative halfword code, both normal and error returns are made to the instruction following the halfword code. In any case, register 15 contains the return code passed back by the called routine.

For OS macro simulation SVC calls and user-handled SVC calls, no error return is recognized by DMSITS. As a result, DMSITS always returns to the calling routine by loading the SVC old PSW that was saved when DMSITS was first entered.

REGISTER RESTORATION: Upon entry to DMSITS, all registers are saved as they were when the SVC instruction was first executed. Upon exiting from DMSITS, all registers are restored to the values that were saved at entry.

The exception to this is register 15 for SVC 202 and 203. Upon return to the calling routine, register 15 contains the value that was in register 15 when the called routine returned to DMSITS after it had completed processing. If the command invoked by the SVC called the parsing facility, any storage allocated by the parsing facility is returned.

Modification of the System Save Area

If the called routine has system status so that it runs with a PSW storage protect key of 0, it may store new values into the system save area.

If the called routine wishes to modify the location where control is to be returned, it must modify the following fields:

- For SVC 202 and 203, the called routine must modify the NUMRET and ERRET (normal and error return address) fields.

- For other SVCs, the called routine must modify the address field of OLDPSW.

To modify the registers that are returned to the calling routine, the fields EGPR1, EGPR2 through EGPR15 must be modified.

If this action is taken by the called routine, the SVCTRACE facility may print misleading information, since SVCTRACE assumes that these fields are exactly as they were when DMSITS was first entered. Whenever an

SVC call is made, DMSITS allocates two save areas for that particular SVC call. Save areas are allocated as needed. For each SVC call, a system and user save area are needed.

When the SVC-called routine returns, the save areas are not released. They are kept for the next SVC. If the routine invoked by the SVC called the parsing facility, any storage allocated by the parsing facility for parsing results is released up on return. At the completion of each command, all SVC save areas allocated by that command are released.

DMSITS uses the system save area (DSECT SSAVE) to save the value of the SVC old PSW at the time of the SVC call, the calling routine's registers at the time of the call, and any other necessary control information. Since SVC calls can be nested, there can be several of these save areas at one time. The system save area is allocated in protected free storage.

The user save area (DSECT EXTUAREA) contains 12 doublewords (24 words) allocated in unprotected free storage. DMSITS does not use this area at all. It simply passes a pointer to this area (via register 13.) The called routine can use this area as a temporary work area or as a register save area. Each system save area has one user save area. The USAVEPTR field in the system save area points to the user save area.

The exact format of the system save area can be found in *VM/SP Data Areas and Control Block Logic Volume 2 (CMS)*. The most important fields and their uses are as follows:

Field **Usage**

CALLER (Fullword) The address of the SVC instruction that resulted in this call.

CALLEE (Doubleword) Eight-byte symbolic name of the called routine. For OS and user-handled SVC calls, this field contains a character string of the form SVC nnn, where nnn is the SVC number in decimal.

CODE (Halfword) For SVC 203, this field contains the halfword code following the SVC instruction line.

OLDPSW (Doubleword) The SVC old PSW at the time that DMSITS was entered.

NRMRET (Fullword) The address of the calling routine where control is passed if there is a normal return from the called routine.

ERRET (Fullword) The address of the calling routine where control is passed if there is an error return from the called routine.

EGPRS (16 Fullwords, separately labeled EGPR0, EGPR1, EGPR2, EGPR3, ..., EGPR15). The contents of the general purpose registers at entry to DMSITS are stored in these fields.

EFPRS (4 Doublewords, separately labeled EFPR0, EFPR2, EFPR4, EFPR6) The entry floating-point registers. The contents of the floating-point registers at entry to DMSITS are stored in these fields.

SSAVENXT (Fullword) The address of the next system save area in the chain. This points to the system save area being used, or will be used, for any SVC call nested in relation to the current one.

SSAVEPRV (Fullword) The address of the previous system save area in the chain. This points to the system save area for the SVC call in relation to which the current call is nested.

USAVEPTR (Fullword) Pointer to the user save area for this SVC call.

Dynamic Linkage/SUBCOM

It is possible for a program that is already loaded from disk to become dynamically known by name to CMS for the duration of the current command; such a program can be called via SVC 202. In addition, this program can also make other programs dynamically known if the first program can supply the entry points of the other programs.

To become known dynamically to CMS, a program or routine invokes the create function of SUBCOM. To invoke SUBCOM, issue the following calling sequence from an assembler language program:

```
         LA  R1,PLIST
         SVC 202
         DC  AL4(ERROR)
         .
         .
         .
PLIST    DS  0F
         DC  CL8'SUBCOM'
SUBCNAME DC  CL8'name'      COMMAND NAME
SUBCPSW  DC  XL2'0000'      SYSTEM MASK, STORAGE KEY, ETC.
         DC  AL2(0)         RESERVED
SUBCADDR DC  A(0)           ENTRY ADDRESS, -1 FOR QUERY PLIST
         DC  A(0)           USER WORD
```

SUBCOM creates an SCBLOCK control block containing the information specified in the SUBCOM parameter list. SVC 202 uses this control block to locate the specified routine. All non-system SUBCOM SCBLOCKS are released at the completion of a command (that is, when CMS displays the ready message). A SUBCOM environment may be defined as a system SUBCOM by setting a X'80' in the first byte of the interruption code field of the PLIST. See *VM/SP Data Areas and Control Block Logic Volume 2 (CMS)* for a description of the SCBLOCK control block.

When a program issues an SVC 202 call to a program that has become known to CMS via SUBCOM, it places X'02' in the high-order byte of register 1. Control passes to the called program at the address specified by the called program when it invoked SUBCOM.

The PSW in the SCBLOCK specifies the system mask, the PSW key to be used, the program mask (and initial condition code), and the starting address for execution. The problem-state bit and machine-check bit may be set. The machine-check bit has no effect in CMS under CP. The EC-mode bit and wait-state bit cannot be set. They are always forced to zero. Also, one 4-byte, user-defined word can be associated with the SUBCOM entry point and referred to when the entry point is subsequently called.

When control passes to the specified entry point, the register contents are:

R2 Address of SCBLOCK for this entry point.

R12 Entry point address.

R13 24-word save area address.

R14 Return address (CMSRET).

R15 Entry point address.

You can also use SUBCOM to delete the potential linkage to a program or routine's SCBLOCK, or you can use SUBCOM to determine if an SCBLOCK exists for a program or routine.

To delete a program or routine's SCBLOCK, issue:

```
DC   CL8'SUBCOM'
DC   CL8'program or routine name'
DC   8X'00'
```

To determine if an SCBLOCK exists for a program or routine, issue:

```
DC   CL8'SUBCOM'
DC   CL8'program or routine name'
DC   A(0) SCBLOCK addressed as a returned value
DC   4X'FF'
```

Note that if 'SUBCOM name' is called from an EXEC file, the QUERY PLIST is the form of PLIST that is issued.

To query the chain anchor, issue:

```
DC   CL8'SUBCOM'
DS   CL8                (contents not relevant)
DS   AL4                Will receive chain anchor
                        contents from NUCSCBLK
DC   AL4(1)             Indicates request for anchor
```

Note that the anchor is equal to F'0' if there are no SCBLOCKs on the chain.

Note: If you create SCBLOCKS for several programs or routines with the same name, they are all remembered, but SUBCOM uses the last one created. A SUBCOM delete request for that name eliminates only the most recently created SCBLOCK making active the next most recently created SCBLOCK with the same name.

When control returns to CMS after a console input command has terminated, the entire SUBCOM chain of SCBLOCKs is released. None of the subcommands established during that command are carried forward to be available during execution of the next console command.

Return Codes

Return codes from the SUBCOM function are:

Return Code	Meaning
0	Successful completion. A new SCBLOCK was created, the specified SCBLOCK was deleted, or the specified program or routine has an SCBLOCK.
1	No SCBLOCK exists for the specified program or routine. This is the return code for a delete or a query.
25	No more free storage available. SCBLOCK cannot be created for the specified program or routine.

Loading and Executing Text Files

The CMS loader consists of a nucleus resident loader (DMSLDR), a file and message handler program (DMSLIO), a library search program (DMSLIB), and other subroutine programs. DMSLDR starts loading at the user first location (AUSRAREA) specified in NUCON or at a user specified location. When performing an INCLUDE function, loading resumes at the next available location after the previous LOAD, INCLUDE, or LOADMOD.

The loader reads in the entire user's program, which consists of one or more control sections, each defined by a type 0 ESD record ("card"). Each control section contains a type 1 ESD card for each entry point and may contain other control cards.

Once the user's program is in storage, the loader begins to search its files for library subprograms called by the program. The loader reads the library subprograms into storage, relocating and linking them as required. To relocate programs, the loader analyzes information on the SLC, ICS, ESD, TXT, and REP cards. To establish linkages, it operates on ESD and RLD cards. Information for end-of-load transfer of control is provided by the END and LDT cards, the ENTRY control card, START command, or RESET option.

The loader also analyzes the options specified on the LOAD and INCLUDE commands. In response to specified options, the loader can:

- Set the load area to zeros before loading (CLEAR option).
- Load the program at a specified location (ORIGIN option).

- Suppress creation of the load-map file on disk (NOMAP option).
- Suppress the printing of invalid card images in the load map (NOINV option).
- Suppress the printing of REP card images in the load map (NOREP option).
- Load program into "transient area" (ORIGIN TRANS option).
- Suppress TXTLIB search (NOLIBE option).
- Suppress text file search (NOAUTO option).
- Execute the loaded program (START option).
- Type the load map (TYPE option).
- Set the program entry point (RESET option).
- Save the relocation information from the text files (RLDSAVE option).
- Save history information (HIST option). However, you must issue the GENMOD command after you issue the INCLUDE or LOAD command with the HIST option.

During its operation, the loader uses a loader table (REFTBL), and external symbol identification table (ESIDTB), and a location counter (LOCCNT). The loader table contains the names of control sections and entry points, their current location, and the relocation factor. (The relocation factor is the difference between the compiler-assigned address of a control section and the address of the storage location where it is actually loaded.) The ESIDTB contains pointers to the entries in REFTBL for the control section currently being processed by the loader. The loader uses the location counter to determine where the control section is to be loaded. Initially, the loader obtains from the nucleus constant area the address (LOCCNT) of the next location at which to start loading. This value is subsequently incremented by the length indicated on an ESD (type 0), END, or ICS card, or it may be reset by an SLC card.

The loader contains a distinct routine for each type of input card. These routines perform calculations using information contained in the nucleus constant area, the location counter, the ESIDTB, the loader table, and the input cards. Other loader routines perform initialization, read cards into storage, handle error conditions, provide disk and typewritten output, search libraries, convert hexadecimal characters to binary, process end-of-file conditions, and begin execution of programs in core. If a card is not one of the recognized types, it is considered a comment card. As a comment card, it can be included in the module by specifying the HIST option on the LOAD or INCLUDE command and then issuing a subsequent GENMOD command.

Following are descriptions of the individual subprocessors with LDR.

SLC Card Routine

Function

This routine sets the location counter (LOCCT) to the address specified on an SLC card or to the address assigned (in the REFTBL) to a specified symbolic name.

Entry

The routine is entered at the first instruction when it receives control from the initial and resume loading routine. It is entered at ORG2 whenever a loader routine requires the current address of a symbolic location specified on an SLC card.

Operation

This routine determines which of the following situations exists, and takes the indicated action:

1. The SLC card does not contain an address or a symbolic name. The SLC card routine branches, via BADCRD in the reference table search routine, to the disk and type output routine (DMSLIO), which generates an error message.

2. The SLC card contains an address only. The SLC card routine sets the location counter (LOCCT) to that address and returns to RD, in the initial and resume loading routine, to read another card.

3. The SLC card contains a name only, and there is a reference table entry for that name. The SLC card routine sets LOCCT to the current address of that name (at ORG2) and returns to the initial and resume loading routine to get another card.

4. The SLC card contains a name only, and there is no reference table entry for that name. The SLC card routine branches via ERRSLC to the disk and type output routine (DMSLIO), which generates an error message for that name.

5. The SLC card contains both an address and a name. If there is a REFTBL entry for the name, the sum of the current address of the name and the address specified on the SLC card is placed in LOCCT. Control returns to the initial and resume loading routine to get another card. If there is no REFTBL entry for the name, the SLC card routine branches via ERRSLC to the disk and type output routine, which generates an error message for the name.

ICS CARD ROUTINE - C2AE1

Function

This routine establishes a reference table entry for the control-segment name on the ICS card if no entry for that name exists, adjusts the location counter to a fullword boundary, if necessary, and adds the card-specified control-segment length to the location counter, if necessary.

Entry

This routine has one entry point, C2AE1. The routine is entered from the initial and resume loading routine when it finds an ICS card.

Operation

1. The routine begins its operation with a test of card type. If the card being processed is not an ICS card, the routine branches to the ESD card analysis routine. Otherwise, processing continues in this routine.

2. The routine tests for a hexadecimal address on the ICS card. If an address is present, the routine links to the DMSLSBA subroutine to convert the address to binary. Otherwise, the routine branches via BADCRD to the disk and type output routine (DMSLIO).

3. The routine next links to the REFTBL search routine, which determines whether there is a reference table entry for the card-specified control-segment name. If such an entry is found, the REFTBL search routine branches to the initial and resume loading routine. Otherwise, the REFTBL search routine places the control-segment name in the reference table and processing continues.

4. The routine determines whether the card-specified control-segment length is zero or greater than zero. If the length is zero, the routine places the current location counter value in the reference table entry as the control segment's starting address (ORG2), and then it branches to the initial and resume loading routine. If the length is greater than zero, the routine sets the current location counter value at a fullword boundary address. The routine then places this adjusted current location counter value in the reference table entry, adjusts the location counter by adding the specified control-segment length to it, and branches to RD in the initial and resume loading routine to get another card.

ESD TYPE 0 CARD ROUTINE - C3AA3

Function

This routine creates loader table and ESID table entries for the card-specified control section.

Entry

This routine has one entry point, C3AA3. The routine is entered from the ESD card analysis routine.

Operation

1. If this is the first section definition, its ESDID is proved.

2. This routine first determines whether a loader table (REFTBL) entry has already been established for the card-specified control section. To do this, the routine links to the REFTBL search routine. The ESD type 0 card routine's subsequent operation

depends on whether there already is a REFTBL entry for this control section. If there is such an entry, processing continues with operation 5, below; if there is not, the REFTBL search routine places the name of this control section in REFTBL and processing continues with operation 3.

3. The routine obtains the card-specified control section length and performs operation 4.

4. The routine links to location C2AJ1 in the ICS card routine and returns to C3AD4 to obtain the current storage address of the control section from the REFTBL entry, inserts the REFTBL entry position (N - where this is the Nth REFTBL entry) in the card-specified ESID table location, and calculates the difference between the current (relocated) address of the control section and its card-specified (assembled) address. This difference is the relocation factor. It is placed in the REFTBL entry for this control section. If previous ESDs have been waiting for this CSECT, a branch is taken to SDDEF, where the waiting elements are processed. A flag is set in the REFTBL entry to indicate a section definition.

5. The entry found in the REFTBL is examined to determine whether it had been defined by a COMMON. If so, it is converted from a COMMON to a CSECT and performs operation 3.

6. If the entry had not been defined previously by an ESD type 0, processing continues at 3.

7. If the entry had been defined previously as other than COMMON, DMSLIO is called via ERRORM to print a warning message, "Duplicate identifier _name.". The entry in the ESID table is set to negative so that the CSECT is skipped (that is, not loaded) by the TXT and RLD processing routines.

ESD TYPE 1 CARD ROUTINE - ENTESD

Function

This routine establishes a loader table entry for the entry point specified on the ESD card, unless such an entry already exists.

Entry

This routine is entered from the ESD card analysis routine.

Operation

1. Branches and links to REFADR to find loader table entry for first section definition of the text deck saved by the ESD 0 routine.

2. The routine then adds the relocation factor and the address of the ESD found in operation 1 or the address in LOCCNT if an ESD has not yet been encountered. The sum is the current storage address of the entry point.

3. The routine links to the REFTBL search routine to find whether there is already a REFTBL entry for the card-specified entry point name. If such an entry exists, the routine performs operation 4. If there is no entry, the routine performs operation 5.

4. Upon finding a REFTBL entry that has been previously defined for the card-specified name, the routine then compares the REFTBL-specified current storage address with the address computed in operation 2. If the addresses are different, the routine branches and links to the DMSLIO routine (duplicate symbol warning); if the addresses are the same, the routine branches to location RD in the initial and resume loading routine to read another card. Otherwise, it is assumed that the REFTBL entry was created as a result of previously encountered external references to the entry. The DMSLSBC routine is called to resolve the previous external references and adjust the REFTBL entry. The entry point name and address are printed by calling DMSLIO.

5. If there is no REFTBL entry for the card-specified entry point name, the routine makes such an entry and branches to the DMSLIO routine.

ESD TYPE 2 CARD ROUTINE - C3AH1

Function

This routine creates the proper ESID table entry for the card-specified external name and places the name's assigned address (ORG2) in the reference table relocation factor for that name.

Entry

This routine has two entry points: C3AH1 and ESD00. Location C3AH1 is entered from the ESD card analysis routine. This occurs when an ESD type 2 card is being processed. Location ESD00 is entered from:

- The ESD card analysis routine, when the card being processed is an ESD type 2 and an absolute loading process is indicated.

- The ESD type 0 card routine and ESD type 1 card routine, as the last operation in each of these routines.

Operation

1. When this routine is entered at location C3AH1, it first links to the REFTBL search routine to determine whether there is a REFTBL entry for the card-specified external name. If none is found, the REFTBL search routine sets the undefined flag for the new loader table entry.

2. The routine resets a possible WEAK EXTRN flag. The routine next places the REFTBL entry's position-key in the ESID table. If the entry has already been defined by means of an ESD type 0, 1,

5, or 6, processing continues at operation 4. Otherwise, it continues at operation 3.

3. The relocated address is placed in the RELFAC entry in the external name's REFTBL entry.

4. The ESD type 2 card routine then determines (at location ESD00) whether there is another entry on the ESD card. If there is another entry, the routine branches to location CA3A1 in the ESD card analysis routine for further processing of this card. Otherwise, the routine branches to location RD in the initial resume loading routine.

Exits

This routine exits to location CA3A1 in the ESD card analysis routine if there is another entry on the ESD card being processed, and it exits to location RD in the initial and resume loading routine if the ESD card requires no further processing.

ESD TYPE 4 ROUTINE - PC

Function

This routine makes loader table and ESIDTAB entries for private code CSECT.

Operation

1. The routine LDRSYM is called to generate a unique character string number of the form 00000001, which is left in the external data area NXTSYM. It is greater in value than the previously generated symbol.

2. The CSECT is then processed as a normal type 0 ESD with the above assigned name.

ESD TYPES 5 AND 6 CARD ROUTINE - PRVESD AND COMESD

Function

This routine creates a reference table and ESIDTAB entries for common and pseudo-register ESDs.

Operation

1. Links to ESIDINC in the ESD type 0 card routine to update the number of ESIDTB entries.

2. Links to the REFTBL search routine to determine whether a reference table (REFTBL) entry has already been created. If there is no entry, the REFTBL search routine places the name of the item in the REFTBL.

3. If the REFTBL search routine had to create an entry for the item, the ESD type 5 and 6 card routine indexes it in the ESIDTB, enters the length and alignment in the entry, indicates whether it

is a PR or common, and branches to ESD00 in the ESD type 2 card routine to determine whether the card contains additional ESDs to be processed. If the entry is a PR, the ESD type 5 and 6 card routine enters its displacement and length in the REFTBL before branching to ESD00.

4. If the REFTBL already contained an entry, the ESD type 5 and 6 card routine indexes it in the ESIDTB, checks alignment, and branches to ESD00.

Note: The PR alignment is coded and placed into the REFTBL. It is an error to encounter more restrictive alignment PR than previously defined. A blank alignment factor is translated to fullword alignment.

ESD TYPE 10 ROUTINE - WEAK EXTRN

The WEAK EXTRN routine calls the search routine to find the EXTRN name in the loader table. If not found, set the WEAK EXTRN flag in the new loader table entry. Exit to ESD00.

TXT CARD ROUTINE - C4AA1

Function

This routine has two functions: address inspection and placing text in storage.

Entry

This routine has three entry points: C4AA1, which is entered from the ESD card analysis routine, and REPENT and APR1, which are entered from the REP card routine for address inspection.

Operation

1. This routine begins its operation with a test of card type. If the card being processed is not a TXT card, the routine branches to the REP card routine. Otherwise, processing continues in this routine.

2. The routine then determines how many bytes of text are to be placed in storage and finds whether the loading process is absolute or relocating. If the loading process is absolute, the routine performs operation 4, below; if relocating, the routine performs operation 3.

3. If the ESIDTB entry was negative, this is a duplicate to CSECT and processing branches to RD. Otherwise, the routine links to the REFADR routine to obtain the relocation factor of the current control segment.

4. The routine then adds the relocation factor (0, if the loading process is absolute) and the card-specified storage address. The result is the address at which the text must be stored. This routine also determines whether the address is such that the text, when loaded starting at that address, overlays the loader or the

reference table. If the loader overlay or a reference table overlay is found, the routine branches to the LDRIO routine. If neither condition is detected, the routine proceeds with address inspection.

5. The routine then determines whether an address has already been saved for possible use as the end-of-load branch address. If an address has been saved, the routine performs operation 7. If not, the routine performs operation 6.

6. The routine determines whether the text address is below location 128. If the address is below location 128, it should not be saved for use as a possible end-of-load branch address, and the routine performs operation 7. Otherwise, the routine saves the address and then performs operation 7.

7. The routine then stores the text at the address specified (absolute or relocated) and branches to location RD in the initial and resume loading routine to read another card.

Exits

The routine exits to two locations:

1. The routine exits to location RD in the initial and resume loading routine if it is being used to process a TXT card.

2. The routine exits to location APRIL in the REP card routine if it is being used for REP card address inspection.

REP CARD ROUTINE - C4AA3

Function

This routine places text corrections in storage.

Entry

This routine has one entry point, C4AA3. The routine is entered from the TXT card routine.

Operation

1. This routine begins its operation with a text of card type. If the card being processed is not a REP card, the routine branches to the RLD card routine. Otherwise, processing continues in this routine.

2. The routine then links to the HEXB conversion routine to convert the REP card-specified correction address from hexadecimal to binary.

3. The routine then links to the HEXB conversion routine again to convert the REP card-specified ESID from hexadecimal to binary.

4. The routine then determines whether the 2-byte correction being processed is the first such correction on the REP card. If it is the

first correction, the routine performs operation 5. Otherwise, the routine performs operation 6.

5. When the routine is processing the first correction, it links to location REPENT in the TXT card routine, where the REP card-specified correction address is inspected for loader overlay and for end-of-load branch address saving. In addition, if the loading process is relocating, the relocated address is calculated and checked for reference table overlay. The routine then performs operation 7.

6. When the correction being processed is not the first such correction on the REP card, the routine branches to location APR1 in the TXT card routine for address inspection.

7. The routine then links to the HEXB conversion routine to convert the correction from hexadecimal to binary, places the correction in storage at the absolute (card-specified) or relocated address, and determines whether there is another correction entry on the REP card. If there is another entry, the routine repeats its processing from operation 4, above. Otherwise, the routine branches to location RD in the initial and resume loading routine.

Exits

When all the REP-card corrections have been processed, this routine exits to location RD in the initial and resume loading routine.

RLD CARD ROUTINE - C5AA1

Function

This routine processes RLD cards, which are produced by the assembler when it encounters address constants within the program being assembled. This routine places the current storage address (absolute or relocated) of a given defined symbol or expression into the storage location indicated by the assembler. The routine must calculate the proper value of the defined symbol or expression and the proper address at which to store that value.

Entry

This routine has two entry points, C5AA1 and PASSTWO.

Operation:

1. Location C5AA1 writes each RLD card into a work file (DMSLDR CMSUT1). Exit to RD to process the next card.

   Location PASSTWO reads an RLD card from the work file. At EOF get to C6AB6 to finish this file.

2. The routine uses the relocation header (RH ESID) on the card to obtain the current address (absolute or relocated) of the symbol referred to by the RLD card. This address is found in the relocation factor section of the proper reference table entry. If the

RHESID is 0, the routine branches to the LDRIO routine (invalid ESD).

3. The routine uses the position header (PH ESID) on the card to obtain the relocation factor of the control segment in which the DEFINE CONSTANT assembler instruction occurred. If the PH ESID is 0, the routine branches to BADCRD in the REFTBL search routine (invalid ESID). If the ESIDTAB entry is negative (duplicate CSECT), the RLD entry is skipped.

4. The routine next decrements the card-specified byte count by 4 and tests it for 0. If the count is now 0, the routine branches to location RD in the initial and resume loading routine. Otherwise, processing continues in this routine.

5. The routine determines the length, in bytes, of the address constant referred to in the RLD card. This length is specified on the RLD card.

6. The routine then adds the relocation factor obtained in operation 3 (relocation factor of the control segment in which the current address of the symbol must be stored) and the card-specified address. The sum is the current address of the location at which the symbol address must be stored.

7. The routine then computes the arithmetic value (symbol address or expression value) that must be placed in storage at the address calculated in operation 6, above, and places that value at the indicated address. If the value is undefined, the routine branches to location DMSLSBB, where the constant is added to a string of constants that are to be defined later.

8. The routine again decrements the byte count of information on the RLD card and tests the result for zero. If the result is zero, go to operation 2. Otherwise, processing continues in this routine.

9. The routine next checks the continuation flag, a part of the data placed on the RLD card by the assembler. If the flag is on, the routine repeats its processing for a new address only--the processing is repeated from operation 4. If the flag is off, the routine repeats its processing for a new symbol--the processing is repeated from operation 2.

Exits

This routine exits to location RD in the initial and resume loading routine.

END CARD ROUTINE - C6AA1

Function

This routine saves the END card address under certain circumstances and initializes the loader to load another control segment.

Entry

This routine has one entry point, C6AA1. The routine is entered from the RLD card routine.

Operation

1. This routine begins its operation with a test of card type. If the card being processed is not an END card, the routine branches to the LDT card routine. Otherwise, processing continues in this routine.

2. The routine then determines whether the END card contains an address. If the card contains no address, the routine performs operation 7, below. Otherwise, the routine performs operation 3.

3. The routine next checks the end-address-saved switch. If this switch is on, an address has already been saved, and the routine performs operation 7. If the switch is off, the routine performs operation 4.

4. The routine determines whether loading is absolute or relocated. If the loading process is absolute, the routine performs operation 6. Otherwise, the routine performs operation 5.

5. The routine links to the REFADR routine to obtain the current relocation factor, and the routine adds this factor to the card-specified address.

6. The routine stores the address (absolute or relocated) in area BRAD for possible use at the end-of-load transfer of control to the problem program.

7. Goes to location PASSTWO (in RLD routine) to process RLD cards.

8. The routine then clears the ESID table, sets the absolute load flag on, and branches to the location specified in a general register (see "Exits").

Exits

This routine exits to the location specified in a general register. This may be either of two locations:

1. Location RD in the initial and resume loading routine. This exit occurs when the END card routine is processing an END card.

2. The location in the LDT card routine, that is specified by that routine's linkage to the END card routine. This exit occurs when

the LDT card routine entered this routine to clear the ESID table and set the absolute load flag on.

CONTROL CARD ROUTINE - CTLCRD1

Function

This routine handles the ENTRY and LIBRARY control cards.

Entry

This routine has one entry point, CTLCRD1. The routine is entered from the LDT card routine.

Operations

1. The CMS function SCAN is called to parse the card.

2. If the card is not an ENTRY or LIBRARY card, the routine determines whether the NOINV option (no printing of invalid card images) was specified. If printing is suppressed, control passes to RD in the initial and resume loading routine, where another card is read. If printing is not suppressed, control passes to the disk and type output routine (DMSLIO), where the invalid card image is printed in the load map. If the card is a valid control card, processing continues.

   ENTRY Card

3. If the ENTRY name is already defined in REFTBL, its REFTBL address is placed in ENTADR. Otherwise, a new entry is made in REFTBL, indicating an undefined external reference (to be resolved by later input or library search), and this REFTBL entry's address is placed in ENTADR.

4. The control card is printed by calling DMSLIO via CTLCRD; it then exits to RD.

   LIBRARY Card

5. Only nonobligatory reference LIBRARY cards are handled. Any others are considered invalid.

6. Each entry-point name is individually isolated and is searched for in the REFTBL. If it has already been loaded and defined, nothing is done and the next entry-point name is processed. Otherwise, the nonobligatory bit is set in the flag byte of the REFTBL entry.

7. Processing continues at operation 4.

REFADR ROUTINE (DMSLDRB)

Function

This routine computes the storage address of a given entry in the reference table.

Entry

This routine has one entry point, REFADR. The routine is entered for several of the routines within the loader.

Operation

1. Checks to see if requested ESDID is zero. If so, uses LOCCNT as requested location and branches to the return location + 44. Otherwise, continues this routine.

2. The routine first obtains, from the indicated ESID table entry, the position (n) of the given entry within the reference table (where the given entry is the nth REFTBL entry).

3. The routine then multiplies n by 16 (the number of bytes in each REFTBL entry) and subtracts this result from the starting address of the reference table. The starting address of the reference table is held in area TBLREF. This address is the highest address in storage, and the reference table is always built downward from that address.

4. The result of the subtraction in operation 2, above, is the storage address of the given reference table entry. If there is no ESD for the entry, goes to operation 5. Otherwise, this routine returns to the location specified by the calling routine.

5. Adds an element to the chain of waiting elements. The element contains the ESD data item information to be resolved when the requested ESDID is encountered.

PRSERCH ROUTINE (DMSLDRD)

Function

This routine compares each reference table entry name with the given name determining (1) whether there is an entry for that name and (2) what the storage address of that entry is.

Entry

This routine is initially entered at PRSERCH and subsequently at location SERCH. The routine is entered from several routines within the loader.

Operation

1. This routine begins its operation by obtaining the number of entries currently in the reference table (this number is contained in area TBLCT), the size of a reference table entry (16 bytes), and

the starting address of the reference table (always the highest address in storage, contained in area TBLREF).

2. The routine then checks the number of entries in the reference table. If the number is zero, the routine performs operation 5. Otherwise, the routine performs operation 3.

3. The routine next determines the address of the first (or next) reference table entry to have its name checked. Increments by one the count it is keeping of name comparisons, and compares the given name with the name contained in that entry. If the names are identical, PRSERCH branches to the location specified in the routine that linked to it. PRSERCH then returns the address of the REFTBL entry. Otherwise, PRSERCH performs operation 4.

4. The routine then determines whether there is another reference table entry to be checked. If there is none, the routine performs operation 5. If there is another, the routine decrements by one the number of entries remaining and repeats its operation starting with operation 3.

5. If all the entries have been checked, and none contains the given name for which this routine is searching, the routine increments by one the count it is keeping of name comparisons, places that new value in area TBLCT, moves the given name to form a new reference table entry, and returns to the calling program.

Exits

This routine exits to either of two locations, both of which are specified by the routine that linked to this routine. The first location is that specified in the event that an entry for the given name is found; the second location is that specified in the event that such an entry is not found.

LOADER DATA BASES

ESD Card Codes (col. 25...)

Code	**Meaning**
00	SD (CSECT or START)
01	LD (ENTRY)
02	ER (EXTRN)
04	PC (Private code)
05	CM (COMMON)
06	XD (Pseudo-register)
0A	WX (WEAK EXTERN)

ESIDTB ENTRY

The ESDID table (ESIDTB) is constructed separately for each text deck processed by the loader. The ESIDTB produces a correspondence between ESD ID numbers (user on RLD cards) and entries in the loader reference table (REFTBL) as specified by the ESD cards. Thus, the ESIDTB is constructed while processing the ESD cards. It is then used to process the TXT and RLD cards in the text deck.

The ESIDTB is treated as an array and is accessed by using the ID number as an index. Each ESIDTB entry is 16 bits long.

Bits	**Meaning**
0	If 1, this entry corresponds to a CSECT that has been previously defined. All TXT cards and RLD cards referring to this CSECT in this text deck should be ignored.
1	If 1, this entry corresponds to a CSECT definition (SDD).
2	Waiting ESD items exist for this ESDID.
3	Unused.
4-15	REFTBL entry number (for example 1, 2, 3, etc.)

Bit 1 is very crucial because it is necessary to use the VALUE field of the REFTBL if the ID corresponds to an ER, CM, or PR; but, the INFO field of the REFTBL entry must be used if the ID corresponds to an SD.

REFTBL ENTRY

0(0) NAME	
8(8) FLAG1	9(9) INFO
12(C) NOTE1	13(D) VALUE
16(10) FLAG2	17(11) ADDRESS

A REFTBL entry is 20 bytes. The fields have the following uses:

NAME
Contains the symbolic name from the ESD data item.

FLAG1

Loader Code	ESD Code	Routine Label	Meaning
7C	00	XBYTE	PR - byte alignment
7D	01	XHALF	PR - halfword alignment
7E	03	XFULL	PR - fullword alignment
7F	07	XDBL	PR - doubleword alignment
80	05	XUNDEF	Undefined symbol
81	04	XCXD	Resolve CXD
82	02	XCOMSET	Define common area
83	05	WEAKEXT	Weak external reference
90	06	CTLLIB	TXTLIBs not to be used to resolve names

INFO

Depends upon the type of the ESD item.

ESD Item Type	INFO Field Meaning
SD (CSECT or START)	Relocation factor
LD (ENTRY)	Zero
CM (COMMON)	Maximum length
PR (Pseudo Register)	

VALUE

Depends upon the type of the ESD item, as does the INFO field.

ESD Item Type	INFO Field Meaning
SD (CSECT or START)	Absolute address
LD (ENTRY)	Absolute address
CM (COMMON)	Absolute address
PR (Pseudo register)	Assigned value (starting from 0)

FLAG2

Bit	Meaning
0	Unused
1	Unused
2	Unused
3	Unused
4	Unused
5	Name was located in a TXTLIB
6	Section definition entry
7	Name specifically loaded from command line

ADDRESS

Unused.

Entries may be created in the loader reference table prior to the actual defining of the symbol. For example, an entry is created for a symbol if it is referenced by means of an EXTRN (ER) even if the symbol has not yet been defined or its type known. Furthermore, COMMON (CM) is not assigned absolute addresses until prior to the start of execution by the START command.

These circumstances are determined by the setting of the flag byte. If the symbol's value has not yet been defined, the value field specifies the address of a patch control block (PCB).

PATCH CONTROL BLOCK (PCB)

These are allocated from free storage and pointed at from REFTBL entries or other PCBs.

Byte	Meaning
0-3	Address of next PCB
4	Flag byte
5-7	Location of ADCON in storage

All address constant locations in loaded program for undefined symbols are placed on PCB chains.

LOADER INPUT RESTRICTIONS

All restrictions that apply to object files for the OS linkage editor apply to CMS loader input files.

Loading and Executing Members of LOADLIBS

The OS relocating loader support consists of two members: the relocating program (DMSLOS) and the overlay program (DMSSFF). In addition, the OSRUN command (DMSOSR) allows the user to invoke directly from the console a program residing in a CMS LOADLIB or an OS module library. DMSOSR executes in user storage.

When a user program invokes the LINK, LOAD, XCTL, or ATTACH SVC, DMSSLN calls DMSLOS to search the libraries in the LOADLIB global list for the specified member name. If found, DMSLOS loads and relocates the requested program from either an OS module library (for example, SYS1.LINKLIB) or a CMS LOADLIB (created by the LKED command). If the member is not found, return is made to DMSSLN to search for a TEXT file or a member of a TXTLIB by that name.

The program exists in the library as text records, directly followed (when required) by control, relocation, and position records. DMSLOS obtains, via the BLDL macro, the information necessary to start loading the program from the PDS directory entry for the program. Then, text records and control records are read alternately, the proper addresses are modified, and return is made to DMSSLN.

The OSRUN command generates a LINK SVC and therefore follows the same path described in the preceding paragraphs. However, if the requested member is not found in searching the libraries specified in the LOADLIB global list, a search is made for a default library ($SYSLIB LOADLIB); TEXT files and TXTLIB members are not searched.

For detailed information on the library record formats, see the *OS/VS Linkage Editor Logic.*

Chapter 8. Manipulating the File System

Part 2 of Figure 8 on page 59 lists the CMS modules that perform either general file system support functions or that perform data manipulation.

Chapter 9. Managing the CMS File System

A description of the structure of the CMS file system and the flow of routines that access and update the file system follows.

Disk Organization

CMS virtual disks (also referred to as minidisks) are blocks of data designed to externally parallel the function of real disks. Several virtual disks may reside on one real disk.

A CMS virtual machine may have up to 26 virtual disks accessed during a terminal session, depending on user specification. Some disks, such as the S-disk, are accessed during CMS initialization. However, most disks are accessed dynamically as they are needed during a terminal session.

How CMS Files are Organized in Storage for an 800-Byte Record

CMS files are organized in storage by three types of data blocks; the file status table (FST), chain links, and file records. Figure 14 on page 116 shows how these types of data blocks relate to each other. The following text and figures describe these relationships and the individual data blocks in more detail.

File Status Tables

CMS files consist of 800-byte records whose attributes are described in the file status table (FST). The file status table is defined by DSECT FSTSECT. The FST consists of such information as the filename, filetype, and filemode of the file, the date on which the file was last written, and whether the file is in fixed-length or variable format. Also, the FST contains a pointer to the first chain link. The first chain link is a block that contains addresses of the data blocks that contain the actual data for the file.

The FSTs are grouped into 800-byte blocks called FST blocks (these are sometimes referred to in listings as hyperblocks). Each FST block contains 20 FST entries, each describing the attributes of a separate file. Figure 15 on page 117 shows the structure of an FST block and the fields defined in the FST.

Note: Programs that modify the fileid of an FST can destroy the integrity of the file system. Programs that modify any of the fields in the FSTs are not supported by CMS. These programs can cause a "file not found" condition for the file until the disk is re-accessed.

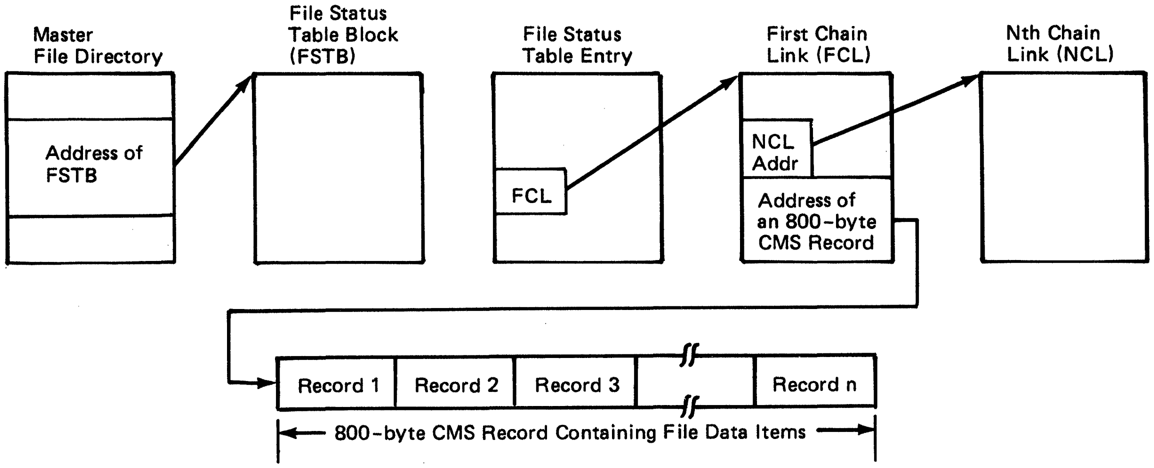


Figure 14. How 800-Byte CMS File Records are Chained Together

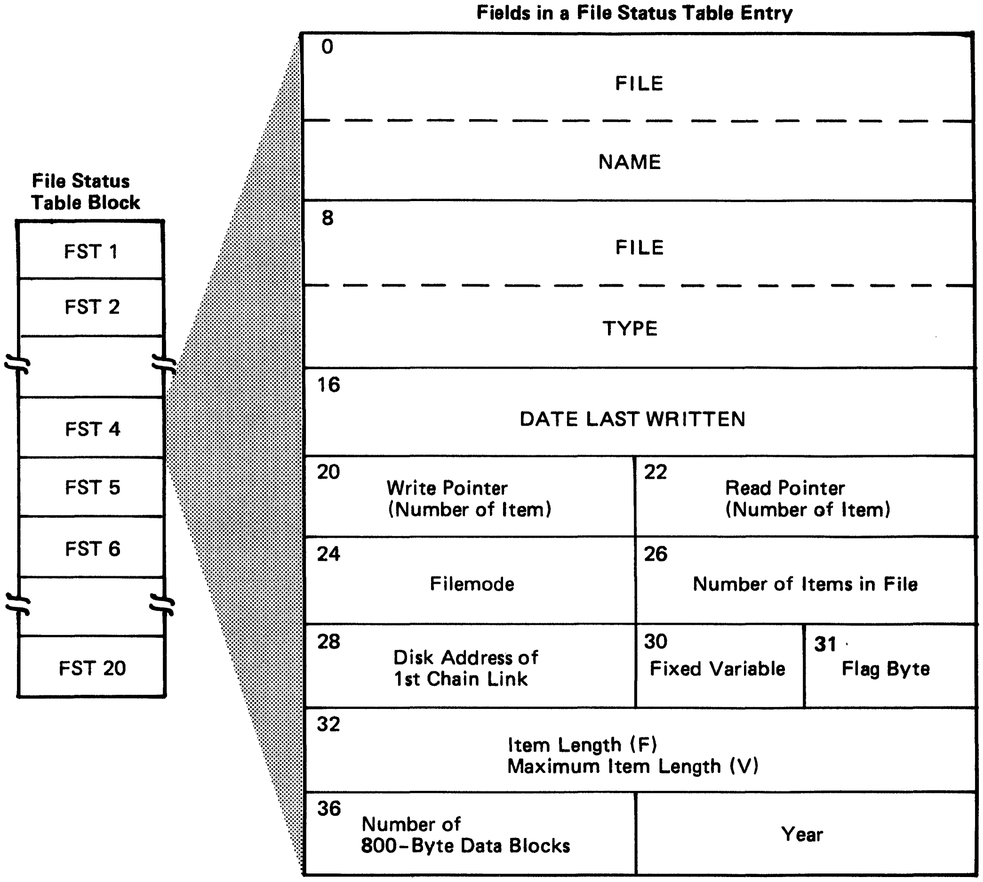


Figure 15. Format of a File Status Table Block - Format of a File Status Table. (for 800-Byte Disk Format)

Chain Links

Chain links are 200- or 800-byte blocks of storage that chain the records of a file in storage. There are two types of chain links: first chain links and Nth chain links.

The first chain link points to two kinds of data. The first 80 bytes of the first chain link contain the halfword addresses of the remaining 40 chain links used to chain the records of the file. The next 120 bytes of the file are the halfword addresses of the first 60 records of the file.

The Nth chain links contain only halfword addresses of the records contained in the file.

Because there are 41 chain links (of which the first contains addresses for only 60 records), the maximum size for any CMS file is 16,060 800-byte records.

CMS Record Formats

CMS records are 800-byte blocks containing the data that comprises the file. For example, the CMS record may contain several card images or ,print images, each is referred to as a record item. Figure 16 shows how chain links are chained together.

CMS records can be stored on disk in either fixed-length or variable-length format. However, the two formats may not be mixed in a single file.

Regardless of their format, the items of a file are stored by CMS in sequential order in as many 800-byte records as are required to accommodate them. Each record (except the last) is completely filled and items that begin in one record can end on the next record. Figure 17 on page 119 shows the arrangement of records in files containing fixed-length records and files containing variable-length records.

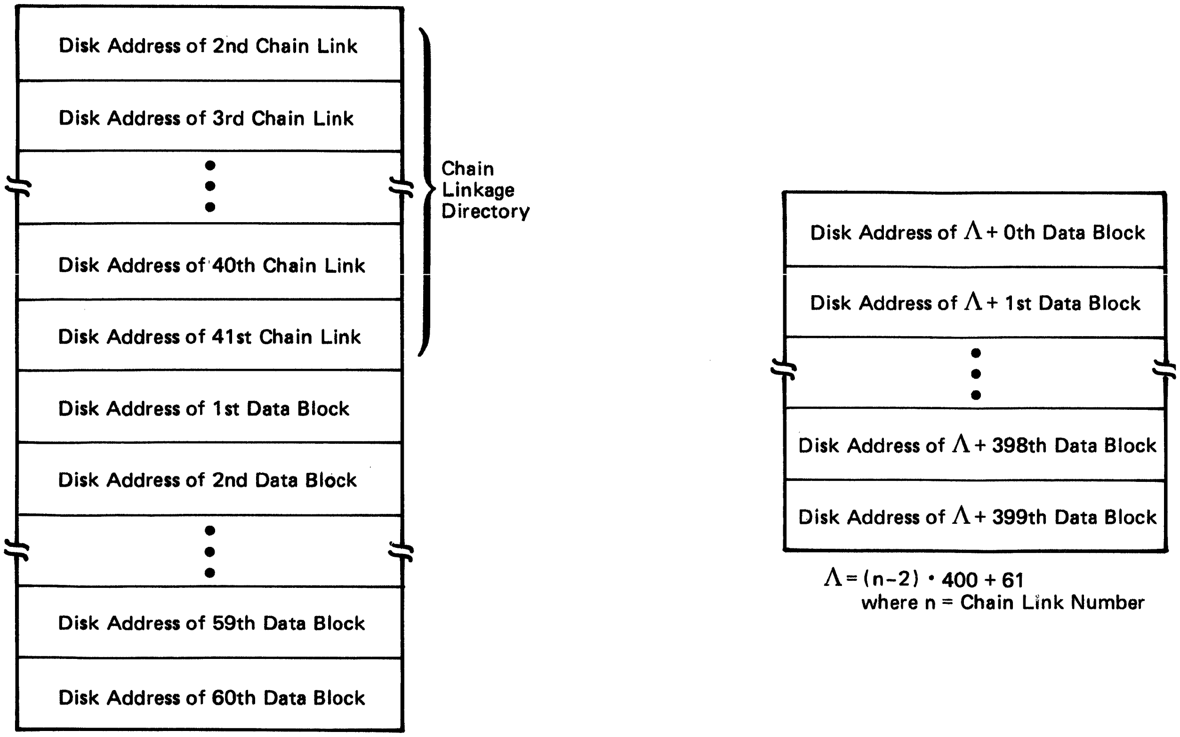
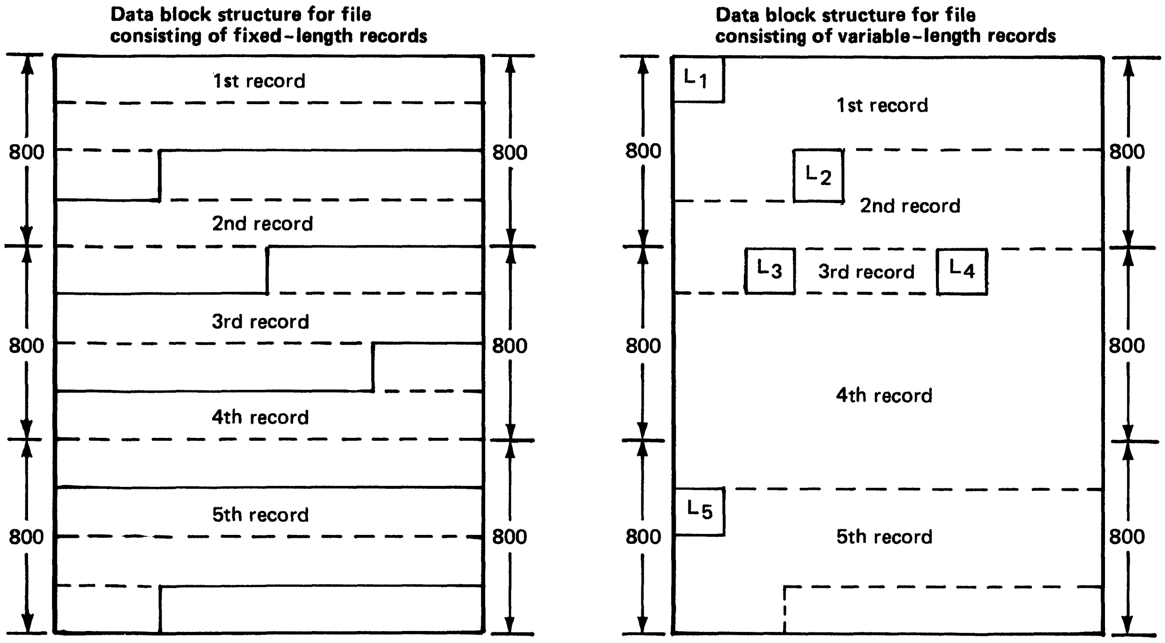

Disk Address of 2nd Chain Link
Disk Address of 3rd Chain Link
Disk Address of 40th Chain Link
Disk Address of 41st Chain Link
Chain Linkage Directory
Disk Address of 1st Data Block
Disk Address of 2nd Data Block
Disk Address of 59th Data Block
Disk Address of 60th Data Block
Disk Address of Λ + 0th Data Block
Disk Address of Λ + 1st Data Block
Disk Address of Λ + 398th Data Block
Disk Address of Λ + 399th Data Block
Λ = (n−2) • 400 + 61
where n = Chain Link Number

Figure 16. Format of the First Chain Link and Nth Chain Links



Data block structure for file consisting of fixed-length records

Data block structure for file consisting of variable-length records

Figure 17. Arrangement of Fixed-Length Records and Variable-Length Records in Files

The location of any item in a file containing fixed-length records is determined by the formula:

```
             (item number - 1) x record length
locations = ---------------------------------
                             800
```

where the quotient is the sequential number of the data block and the remainder is the displacement of the item into the data block.

For variable-length records, each record is preceded by a 2-byte field specifying the length of the record.

Physical Organization of Virtual Disks

Virtual disks are physically organized in 800-byte records. Records 1 and 2 of each user disk are reserved for IPL. Record 3 contains the disk label. Record 4 contains the master file directory. The remaining records on the disk contain user file-related information such as the FSTs, chain links, and the individual file records discussed above.

The Master File Directory

The master file directory (MFD) is the major file management table for a virtual disk. As mentioned earlier, it resides on cylinder 0, track 0, record 4 of each virtual disk. The master file directory contains six types of information.

- The disk addresses of the FST entries describing user files on that disk.

- A 4-byte "sentinel," which can be either FFFD or FFFF. FFFD specifies that extensions of the QMSK (described below) follow. FFFF specifies that no QMSK extension follow.

- Extensions to the QMSK, if any.

- General information describing the status of the disk:

  - ADTNUM — The total number of 800-byte blocks on the user's disk.

  - ADTUSED — The number of blocks currently in use on the disk.

  - ADTLEFT — Number of blocks remaining for use (ADTNUM - ADTUSED).

  - ADTLAST — Relative byte address of the last record in use on the disk.

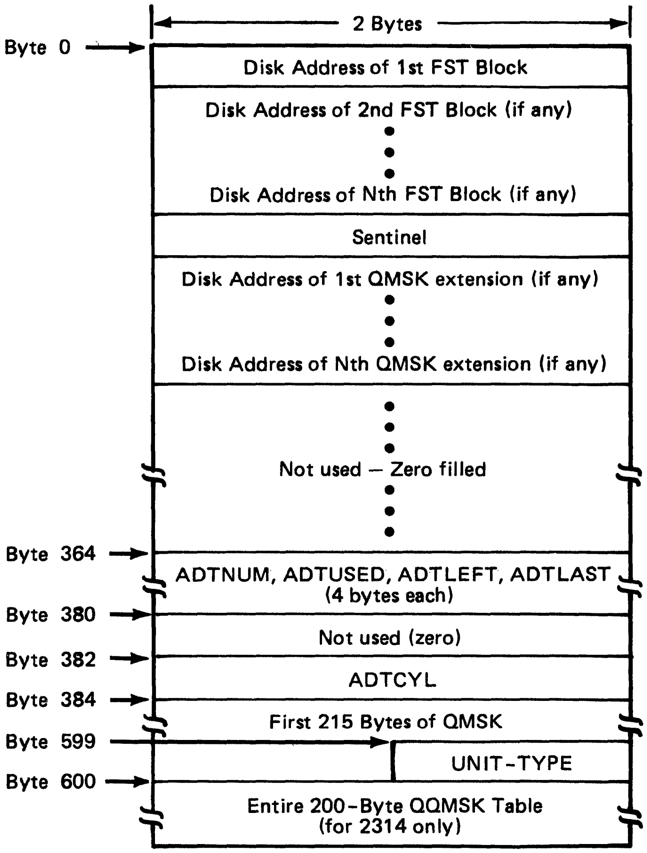
  - ADICYL — Number of cylinders on the user's disk.

  - Unit Type — A 1-byte field describing the type of the disk: 07 for a 3340, 08 for a 2314, 09 for a 3330, 0B for a 3350, 0C for a 3375, 0E for a 3380, FE for a 3370, and FF for a 3310.

  - A bit mask called the QMSK, which keeps track of the status of the records on disk.

  - Another bit map, called the QQMSK, which is used only for 2314 disks and performs a function similar to that of QMSK.

Figure 18 on page 121 shows the structure of the master file directory. Figure 14 on page 116 shows the relationship of the master file directory, which resides on disk, to data blocks brought into storage for file management purposes, for example, FSTs and chain links.



2 Bytes

Byte 0 → Disk Address of 1st FST Block

Disk Address of 2nd FST Block (if any)
•
•
•
Disk Address of Nth FST Block (if any)

Sentinel

Disk Address of 1st QMSK extension (if any)
•
•
•
Disk Address of Nth QMSK extension (if any)

•
•
•
Not used – Zero filled
•
•
•

Byte 364 → ADTNUM, ADTUSED, ADTLEFT, ADTLAST (4 bytes each)

Byte 380 → Not used (zero)

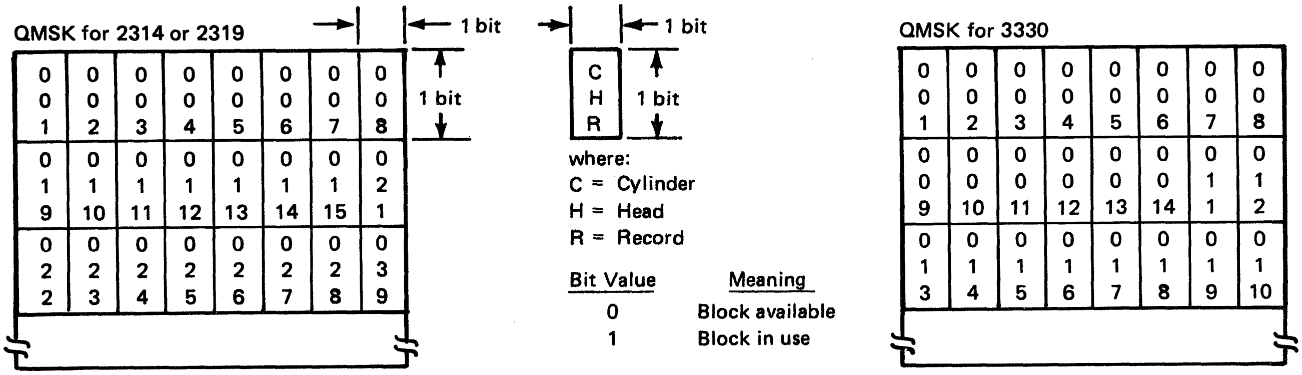
Byte 382 → ADTCYL

Byte 384 → First 215 Bytes of QMSK

Byte 599 → UNIT-TYPE

Byte 600 → Entire 200-Byte QQMSK Table (for 2314 only)

Figure 18. Structure of the Master File Directory



Number of QMSK Extensions	Number of Cylinders on Disk			
Required (if any)	2314 or 2319	3330	3340	3350
0	1 – 11	1 – 6		
1	12 – 54	7 – 30		
2	55 – 96	31 – 54		
3	97 – 139	55 – 78		
4	140 – 182	79 – 102		
5	183 – 203	103 – 126		
6	–	127 – 150		
7	–	151 – 174		
8	–	175 – 198		
9	–	199 – 223		
10	–	224 – 246		

Figure 19. Disk Storage Allocation Using the QMSK Data Block

Keeping Track of Read/Write Disk Storage: QMSK and QQMSK

Because CMS does not require contiguous disk space, disk space management needs to determine only the availability of 800-byte blocks and to chain them together. The status of the blocks on any read/write disk (which blocks are available and which are currently in use) is stored in a table called QMSK. The term QMSK is derived from the fact that a 2311 disk drive has four 800-byte blocks per track. One block is a "quarter-track", or QTRK, and a 200-byte area is a "quarter-quarter-track", or QQTRK. The bit mask for 2314, 2319, 3310, 3330, 3340, 3350, 3370, 3375, or 3380 records is called the QMSK, although each 800-byte block represents less than a quarter of a track on these devices.

On a 2314 or 2319 disk, the blocks are actually grouped fifteen 800-byte blocks per even/odd pair of tracks. An even/odd pair of tracks is called a track group. On a 3330 disk, the blocks are grouped fourteen 800-byte blocks per track. On a 3340 disk, the blocks are grouped into eight 800-byte blocks per track.

When the system is not in use, a user's QMSK resides on the master file directory. During a session it is maintained on disk, but also resides in main storage. QMSK is of variable length, depending on how many cylinders exist on the disk.

Each bit is associated with a particular block on the disk. The first bit in QMSK corresponds to the first block, the second bit to the second block, and so forth, as shown in Figure 19 on page 122.

When a bit in QMSK is set to 1, it indicates that the corresponding block is in use and not available foe allocation. A 0-bit indicates that the corresponding block is available. The data blocks are referred to by relative block numbers throughout disk space management, and the disk I/O routine, DMSDIO, finally converts this number to a CCHHR disk address.

A table called QQMSK indicates which 200 byte segments (QQTRK) are available for allocation and which are currently in use. QQMSK contains 100 entries, which are used to indicate the status of up to 100 QQTRK records. An entry in QQMSK contains either a disk address, pointing to a QQTRK record that is available for allocation, or zero. QQMSK is used only for 2314 files; for 3330, 3340, and 3350, the first chain link occupies the first 200-byte area of an 800-byte block.

The QMSK and QQMSK tables for read-only disks are not brought into storage, since no space allocation is done for a disk while it is read-only. They remain, as is, on the disk until the disk is accessed as a read/write disk.

Dynamic Storage Management: Active Disks and Files

CMS disks and files contained on disk are physically mapped using the data blocks described above: for disks, the MFD, the QMSK, and the QQMSK; for files, the FST, chain links, and 800-byte file records. In storage, all of this data is accessed by means of two DSECTs whose addresses are defined in the DSECT NUCON, ADTSECT and AFTSECT.

Managing Active Disks: The Active Disk Table

The ADTSECT DSECT maps information in the active disk table (ADT). This information includes data contained in the MFD, FST blocks, the QMSK, and QQMSK. The DSECT comprised of ten "slots," each representing one CMS virtual disk. A slot contains significant information about the disk such as a pointer to the MFD for the disk, a pointer to the first FST block, and pointers to the QMSK and QQMSK, if the disk is a R/W disk. ADTSECT also contains information such as the number of cylinders on the disk and the number of records on the disk.

Managing Active Files: The Active File Table

Each open file is represented in storage by an active file table (AFT). The AFT (defined by the AFTSECT DSECT) contains data found on disk in FSTs, chain links, and data records. Also contained in the AFT is information such as the address of the first chain link for the file, the current chain link for the file, the address of the current data block and the fileid information for the file. Figure 2 on page 8 shows the relationship between the AFT and other CMS data blocks.

CMS Routines Used to Access the File System

DMSACC is the control routine used to access a virtual disk. In conjunction with DMSACM and DMSACF, DMSACC, DMSACP, and DMSACS build, in virtual storage, the tables CMS requires for processing files contained on the disk. The list below shows the logical flow of the main function of DMSACC.

Access a Virtual Disk: DMSACC

DMSACC
Scans the command line to determine which disk is specified.

DMSLAD
Looks up the address of the ADT for the disk specified on the command line.

DMSACC
Determines whether an extension to a disk has been specified on the command line and ensures that it is correctly specified.

DMSLAD
In the case where an extension has been specified, ensures that the extension disk exists.

DMSLAD
Ensures that the specified disk is not already accessed as a R/W disk.

DMSFNS
In the case where the specified disk is replacing a currently accessed disk, closes any open files belonging to the duplicate disk.

DMSACC
Verifies the parameters remaining on the command line.

DMSALU
Releases any free storage belonging to the duplicate disk via a call to DMSFRE. Also, clears appropriate entries in the ADT for use by the new disk.

DMSACM
(Called as the first instruction by DMSACF) Reads from the Master File Directory, the QMSK, and the QQMSK for the specified disk. Also, DMSACM updates the ADT for the specified disk using information form the MFD.

DMSACF
Reads into storage all the FST blocks associated with the specified disk. DMSACF calls DMSHTB to build hyperblock mapping tables for read/only disks (if the hyperblocks that are searched span three or

more pages). DMSACF also calls DMSHTB to build hash tables for read/write EDF disks (if the hyperblocks that are searched span two or more pages).

DMSACF calls DMSACG for EDF disks that are not S- or Y-disks to read in the directory by hyperblocks and to sort, if necessary.

DMSACG
If sufficient storage is available and if the disk in question is EDF but not an S- or Y- disk, DMSACG is called by DMSACF. The directory is read into contiguous storage by hyperblocks and sorted, if necessary, bypassing the call to DMSALU (SORTFST). Control is then returned to DMSACF.

DMSACC
Handles error processing or processing required to return control to DMSINT.

How CMS Files are Organized in Storage for 512-, 1K-, 2K-, or 4K-byte Records on Disk

CMS files are organized by three types of blocks; the file status table (FST), pointer blocks, and file records. Figure 20 on page 126 shows how these types of blocks relate to each other. The following text and figures describe these relationships and the individual data blocks in more detail.

File Status Tables

CMS files consists of 512-, 1K-, 2K-, or 4K-byte CMS blocks whose attributes are described in the file status table (FST). The file status table is defined by DSECT FSTSECT. The FST consists of such information as the filename, filetype, and filemode of the file, the date on which the file was last written, and whether the file is in fixed-length or variable format. Also, the FST contains a pointer to the highest level pointer block or only data block. If it is a pointer block, this block contains addresses of the next lower level pointer blocks or the data blocks that contain the actual data for the file.

The FSTs are grouped into 512-, 1K-, 2K-, or 4K-byte CMS blocks called FST blocks (these are sometimes referred to in listings as hyperblocks). Each FST block contains 8, 16, 32, or 64 FST entries respectively (an FST is 64 bytes long), each describing the attributes of a separate file. Figure 21 on page 127 shows the structure of an FST block and the fields defined in the FST.

Note: Programs that modify the fileid of an FST can destroy the integrity of the file system. Programs that modify any of the fields in the FSTs are not supported by CMS. These programs can cause a "file not found" condition for the file until the disk is re-accessed.

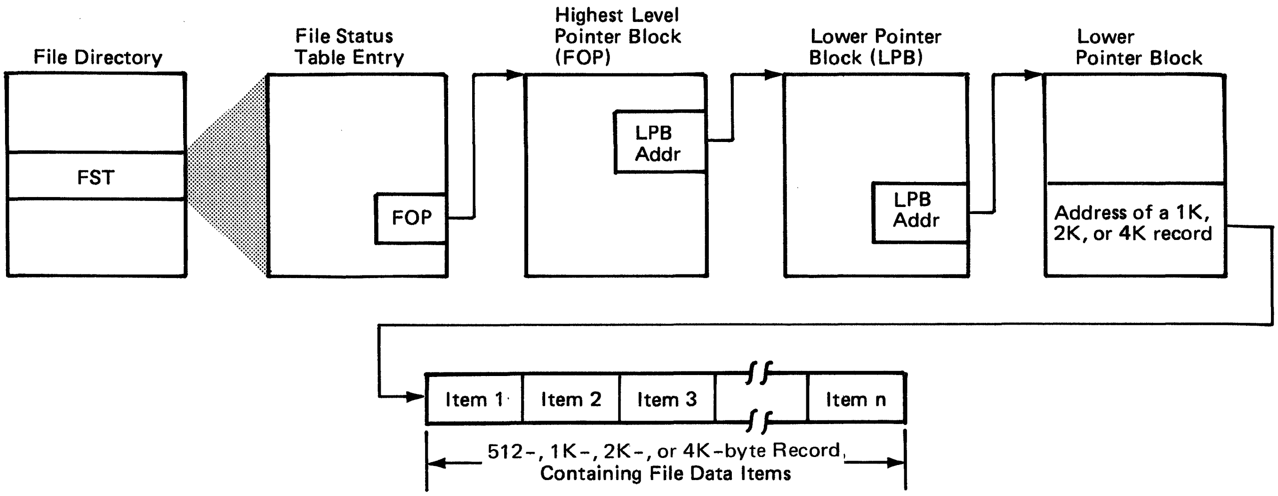
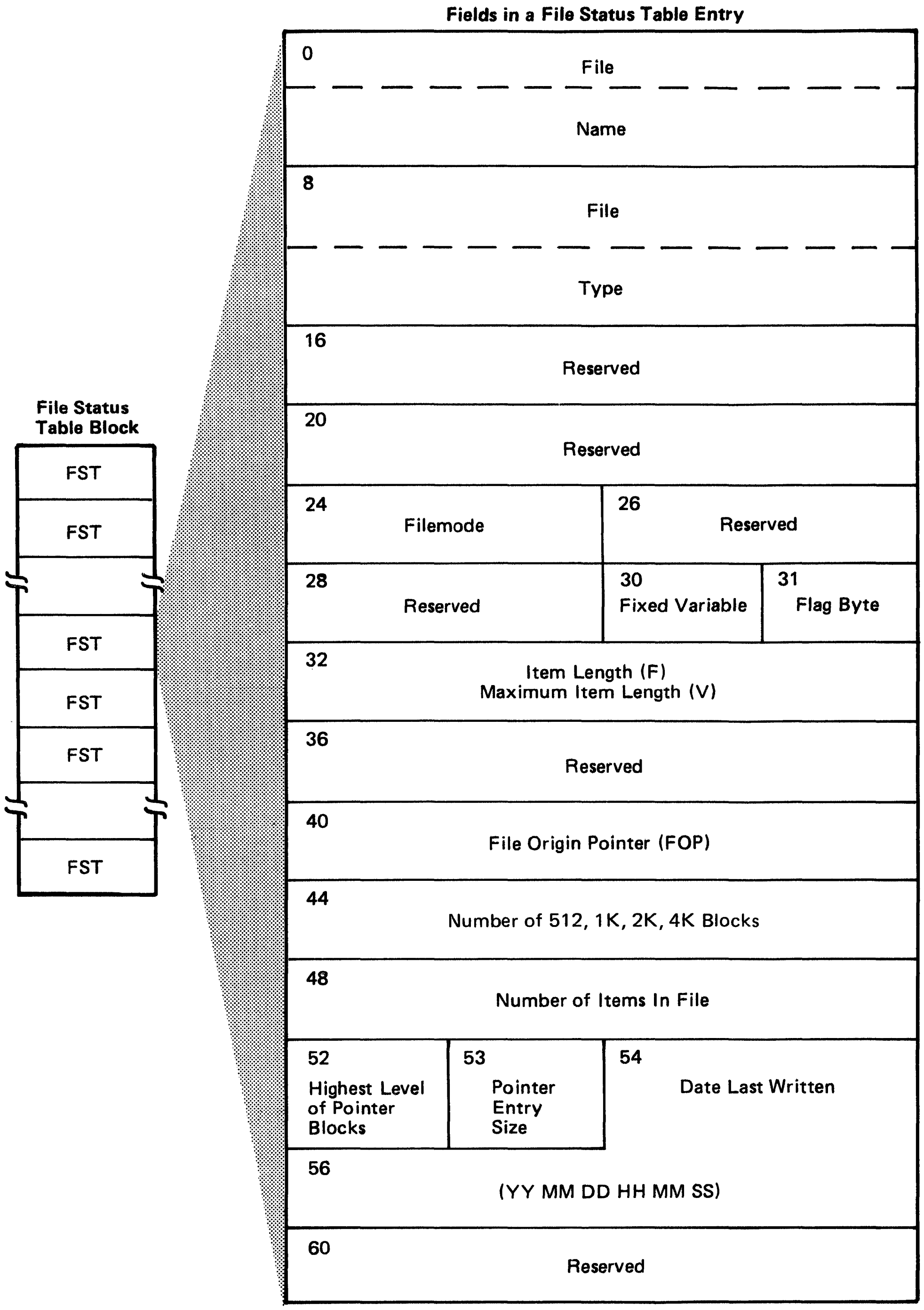


Figure 20. How 512-, 1K-, 2K-, or 4K-Byte CMS File Records are Chained Together



Fields in a File Status Table Entry

0	File Name		
8	File Type		
16	Reserved		
20	Reserved		
24	Filemode	26 Reserved	
28	Reserved	30 Fixed Variable	31 Flag Byte
32	Item Length (F) Maximum Item Length (V)		
36	Reserved		
40	File Origin Pointer (FOP)		
44	Number of 512, 1K, 2K, 4K Blocks		
48	Number of Items In File		
52	Highest Level of Pointer Blocks	53 Pointer Entry Size	54 Date Last Written
56	(YY MM DD HH MM SS)		
60	Reserved		

Figure 21. Format of a File Status Table Block - Format of a File Status Table. (For 512-, 1K-, 2K-, 4K-Byte Disk Format)

Pointer Blocks

Pointer blocks are 512-, 1K-, 2K-, or 4K-byte blocks of storage that chain the records of a file. There are up to five levels of pointer blocks. All but the first level of pointer blocks contain the fullword disk address of the next lower level pointer block. The level-one pointer blocks contain the fullword disk addresses of the data blocks of the file (see Figure 22 on page 129 and Figure 23 on page 130).

There are two types of pointer blocks: pointer blocks for fixed files which are as described above, and pointer blocks for variable files. For the variable files, each pointer block entry is three fullwords long. The first fullword holds the disk address of the next lower level pointer block, the next fullword holds the highest item number contained in this lower corresponding pointer block, and the last fullword holds the displacement, at the data level, to the first identified item contained in a lower corresponding pointer block. CMS blocks are not shared by files.

Each entry of a level-one pointer block is composed of one fullword containing the disk address of the corresponding data block, one fullword containing the highest item number contained in this data block, and one fullword containing the displacement, in bytes, of the first identified item (if any) contained in this data block. This last fullword of the entry may hold the hexadecimal value X'FF...F', indicating that the item is spanned.

The last fullword of a pointer block holds the displacement, in bytes, of the last used entry, if one exists, in the block. This structure permits the creation of very large files. The maximum number of data blocks available in a variable format file on a 1K-, 2K-, or 4K-byte blocksize minidisk is about 2^{31} - 1. The maximum number of data blocks available in a variable format file on a 512-byte blocksize minidisk is about 15 times less than 2^{31} - 1. The maximum number of blocks available in a variable format file is 64K.

Each pointer block or data block is prefixed in virtual storage with a header. This header holds an entry called DCHTRUNK that points to the upper level pointer block. Associated with the DCHTRUNK value is a displacement that indicates the corresponding entry in this upper level pointer block.

In virtual storage, each level of pointer block and the data block have an anchor in the corresponding active file table (AFT) and are forward and backward chained by the prefix.

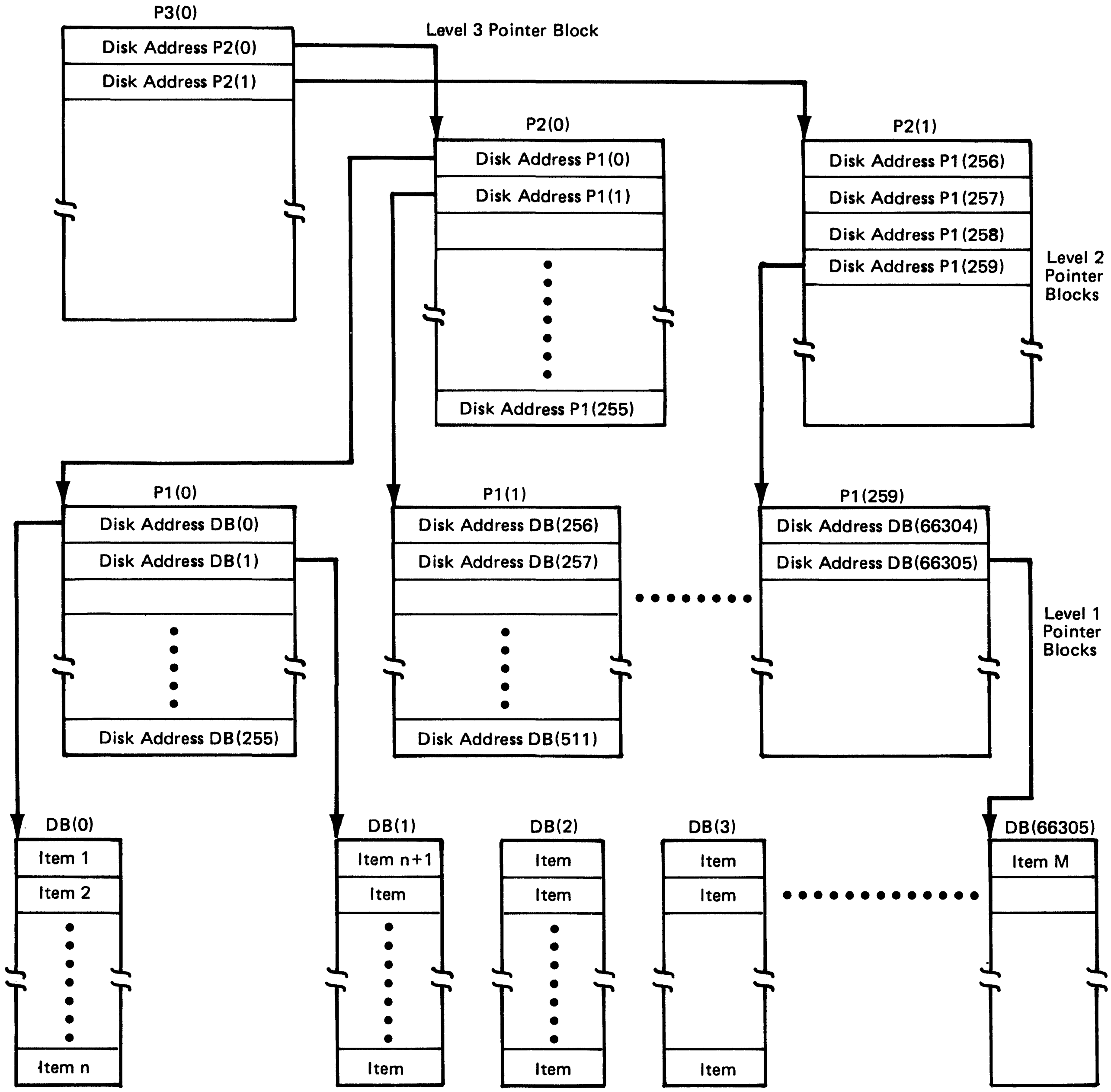


Figure 22. Format of Level 3 Pointer Block Fixed-Length Record File

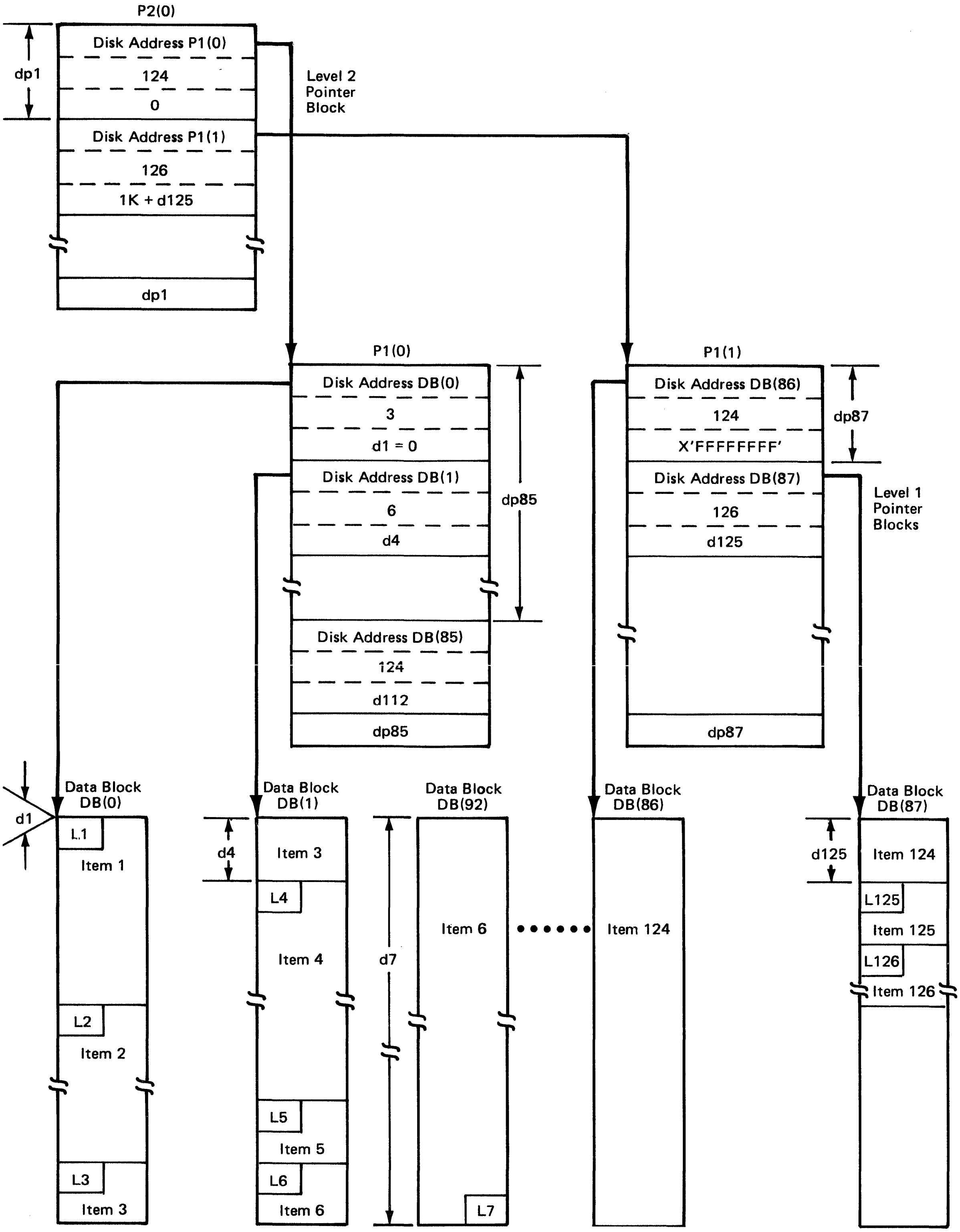


Figure 23. Format of Level 2 Pointer Block Variable-Length Record File

CMS Block Formats

CMS blocks are 512-, 1K-, 2K-, or 4K-byte disk records containing the data that comprises the file. For example, the CMS record may contain several card images or print images, each of which is referred to a record item. Figure 22 on page 129 show how pointer blocks are chained together.

CMS file items can be stored on disk in either fixed-length or variable-length format. However, the two formats may not be mixed in a single file.

Regardless of their format, the items of a file are stored by CMS in sequential order in as many 512-, 1K-, 2K-, or 4K-byte records as are required to accommodate them. Each CMS block (except the last) is completely filled and items that begin in one CMS block can end in the next CMS block. Figure 22 on page 129 shows the arrangement of items in files containing fixed-length items and files containing variable-length items.

The location of any item in a file containing fixed-length items is determined by the formula:

```
           (item number - 1) x record length
location = ---------------------------------
                  512, 1K, 2K, or 4K
```

where the quotient is the sequential number of the data block and the remainder is the displacement of the item into the data block.

For variable-length files, each item is preceded by a 2-byte field specifying the length of the item.

Physical Organization of Virtual Disks

Virtual disks are physically organized in 512-, 1K-, 2K-, or 4K-byte disk records. Records 1 and 2 of each user disk are reserved for IPL. Record 3 contains the disk label. The first block of the file directory is alternately exchanged between record 4 and record 5 when the directory is rewritten to disk. The remaining records on the disk contain information such as allocation map blocks, FSTBs, pointer blocks, and the individual file records as discussed above.

CMS disk structures that reside on FB-512 devices are 512-, 1024-, 2048-, or 4096-byte CMS block format. The required number of 512-byte physical FB-512 disk records are logically concatenated together to form each CMS block. For example; on a 1024-byte format disk, FB-512 physical record numbers 0 and 1 (origin 0) are used together to form CMS block 1 (origin 1). The FB-512 label occupies FB-512 block 1 (origin 0) leaving CMS blocks 2 and 3 available for general use.

The File Directory, the Allocation Map, and the Disk Label

The file directory and the allocation map have the same organization as files. The directory contains FSTs and the first block resides on cylinder 0, track 0, record 4 or record 5 of each virtual disk. The record number (4 or 5) is maintained in the field disk origin pointer of the disk label.

The directory itself is described by an FST that is the first FST in the first block. The filename for the directory is binary zero (except for byte 4 which is binary 1), and the filetype is "DIRECTOR".

The allocation map is described by an FST that is the second FST in the first block of the directory. The filename is binary zero (except for byte 4 which is binary 2), and the filetype is "ALLOCMAP".

The disk label resides on cylinder 0, track 0, record 3. It is 80-byte long and contains the following information:

ADTIDENT CMS1 is the label identifier.

ADTID Six characters given by the user are the volume identifier.

ADTDBSIZ One fullword; contains the disk block size that the user chooses at format disk time (512, 1K, 2K, or 4K).

ADTDOP One fullword; contains records 4 or 5 depending upon the actual directory first data block address.

ADTCYL One fullword; contains the number of formatted cylinders available for CMS files.

ADTMCYL One fullword; contains the maximum number of formatted cylinders, that is, the size of the disk.

ADTNUM One fullword; the total number of 512-, 1K-, 2K-, or 4K-byte blocks on the user's disk.

ADTUSED One fullword; the number of blocks currently in use on the disk.

ADTFSTSZ One fullword; the size of the FST (64 bytes).

ADTNFST One fullword; the number of FSTs per block.

ADTCRED Six characters; the disk creation date (YYMMDDHHMMSS).

ADTAMNB One fullword; contains the relative block number of the current cursor position within the allocation map.

ADTAMND One fullword; contains the relative byte offset of the current cursor position within the block specified by ADTAMNB.

ADTAMUP	One fullword; contains the relative byte address within the allocation map that corresponds to the start of user data blocks.
ADTSFNAM	Eight characters; contains the name of the discontiguous shared segment (DCSS) if a saved file directory is to be used for this disk.

Keeping Track of Read/Write Disk Storage: Allocation Map

In CMS, disk space is composed of 512-, 1K-, 2K-, or 4K-byte blocks chained together. Because disk space management only determines the availability of blocks, not extents, it need not allocate disk space contiguously. The status of the blocks on any read/write disk (which blocks are available and which are currently in use) is stored in a table called the allocation map. The allocation map contains bits, each of which is associated with a particular CMS block. The first corresponds to the first CMS block, the second bit corresponds to the second CMS block, and so forth.

When a bit in the allocation map is set to 1, it indicates that the corresponding block is in use and not available for allocation. A 0-bit indicates that the corresponding block is available. The data blocks are referred to by relative block numbers through disk space management, and the disk I/O routine, DMSDIO, finally converts this number to a CCHHR disk address or FB-512 block number.

When the system is not in use, a user's allocation may reside on the corresponding disk. During a session, it is maintained on disk but also resides in real storage. The allocation map is variable in length, depending on how many cylinders exist on the disk. The CMS disk may reside on the entire physical disk pack and is limited only by the physical limit of the disk pack.

A deallocation map exists in real storage when CMS disk blocks are deallocated. During a terminal session, a block is recorded as deallocated by turning on its corresponding bit in the deallocation map.

When the disk is updated by rewriting the file directory and the allocation map, the current allocation map is formed by combining the allocation map and the deallocation map. In fact, a deallocation map block is created only for those allocation map blocks in which a CMS block is deallocated.

The allocation maps for read-only disks are not brought into storage because no space allocation is performed for a disk while it is in read-only status. They remain, as is, on the disk until the disk is accessed as a read-write disk.

Selective Directory Update

The file directory and the allocation map are built with CMS blocks (512-, 1K-, 2K-, or 4K-bytes). The selective directory update function takes place when the file directory and the allocation map must be updated on the corresponding disk. It writes on disk only the modified blocks of the directory (including required pointer blocks) and the entire allocation map.

Dynamic Storage Management: Active Disks and Files

CMS disks are physically mapped in CMS blocks containing the file directory and the allocation map. CMS files on disk are mapped using FST blocks, pointer blocks, and 512-, 1K-, 2K-, or 4K-byte file data blocks.

In real storage all of this data is accessed by means of two DSECTs whose addresses are defined in DMSNUC, ADTSECT, and AFTSECT. 10 ADTSECTs reside in DMSNUC and the others (11 through 26) reside in free storage when they are used. Five AFTs reside in DMSNUC and the others reside in free storage. (See Figure 24 on page 135).

Managing Active Disks: The Active Disk Table

The ADTSECT DSECT maps information in the active disk table (ADT). An ADT contains significant information about the CMS disk such as the anchors for pointer block levels, the data block for the file directory, and the data block for the allocation map (if the disk is a read-write disk). The ADTSECT also contains disk label information.

Managing Active Files: The Active File Table

Each open file is represented in storage by an active file table (AFT). The AFT (defined by AFTSECT DSECT) contains data found on disk in FSTs, the anchors for pointer block levels and the data block for the file. The AFT also contains such information as the read pointer and write pointer of the file, the number of entries in a pointer block, the number of pointer block levels, and the length of a pointer block entry. Figure 24 on page 135 shows the relationship between the AFT and other CMS blocks.

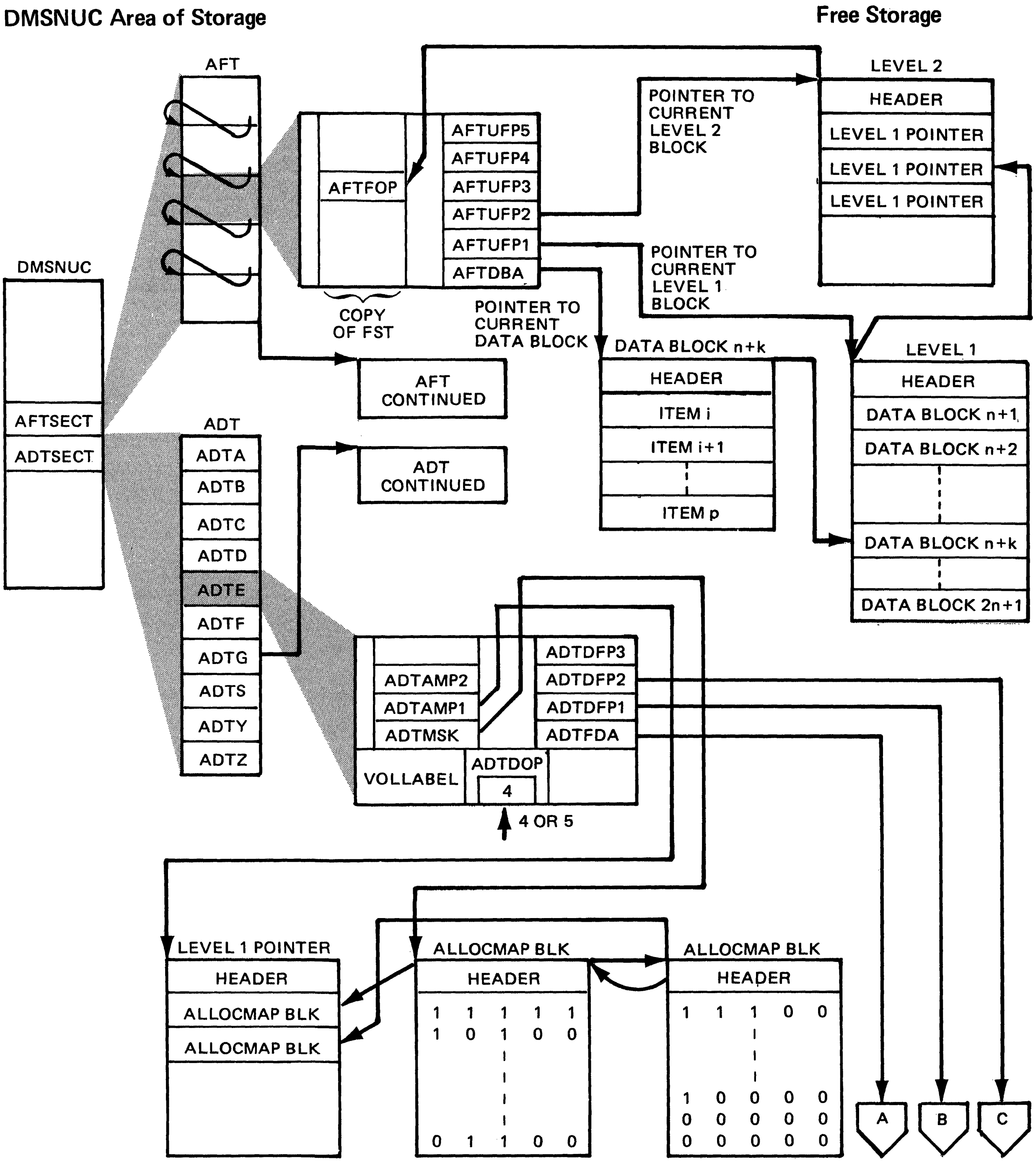


Figure 24 (Part 1 of 3). File System for 512-, 1K-, 2K-, or 4K-Byte Record on Disk

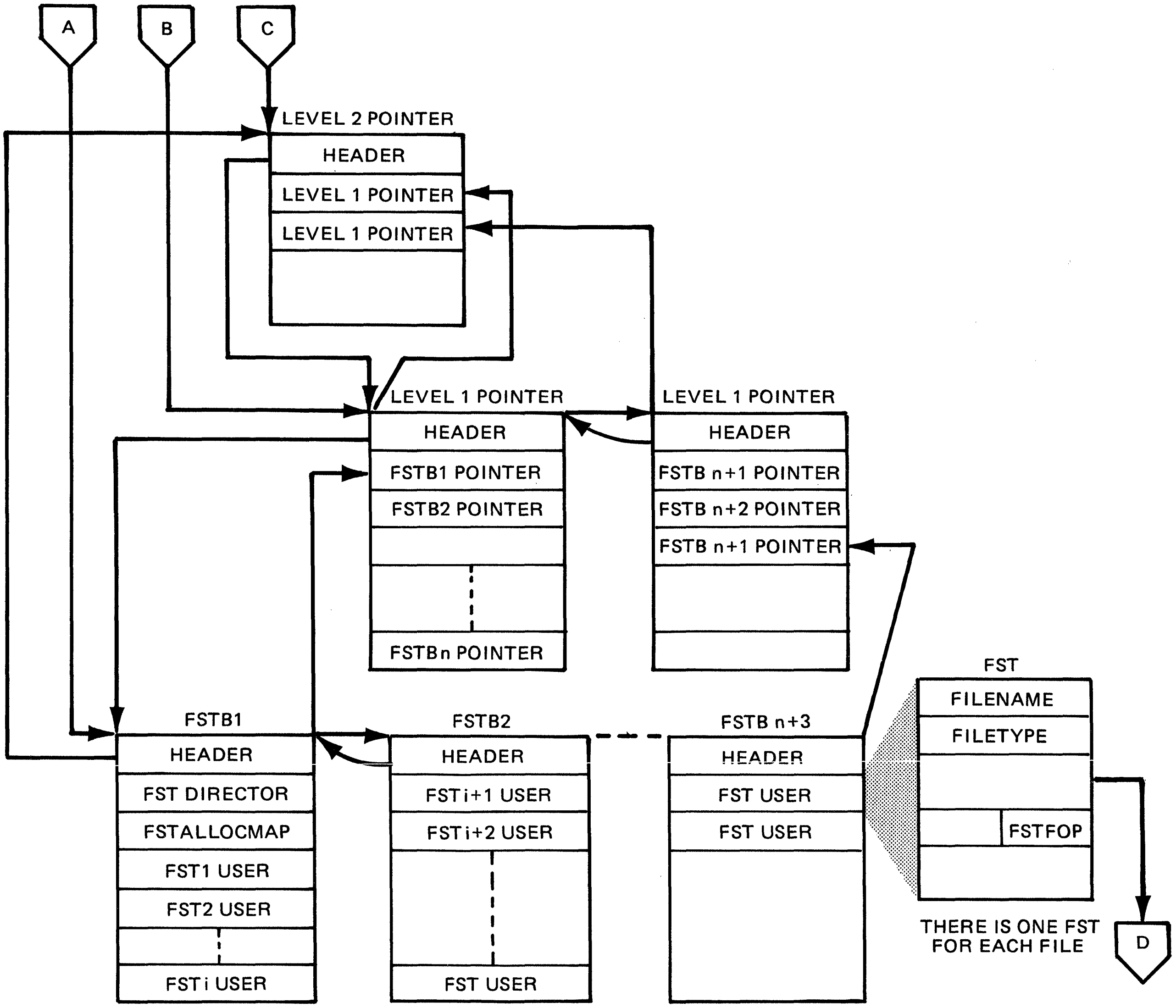


Figure 24 (Part 2 of 3). File System for 512-, 1K-, 2K-, or 4K-Byte Record on Disk

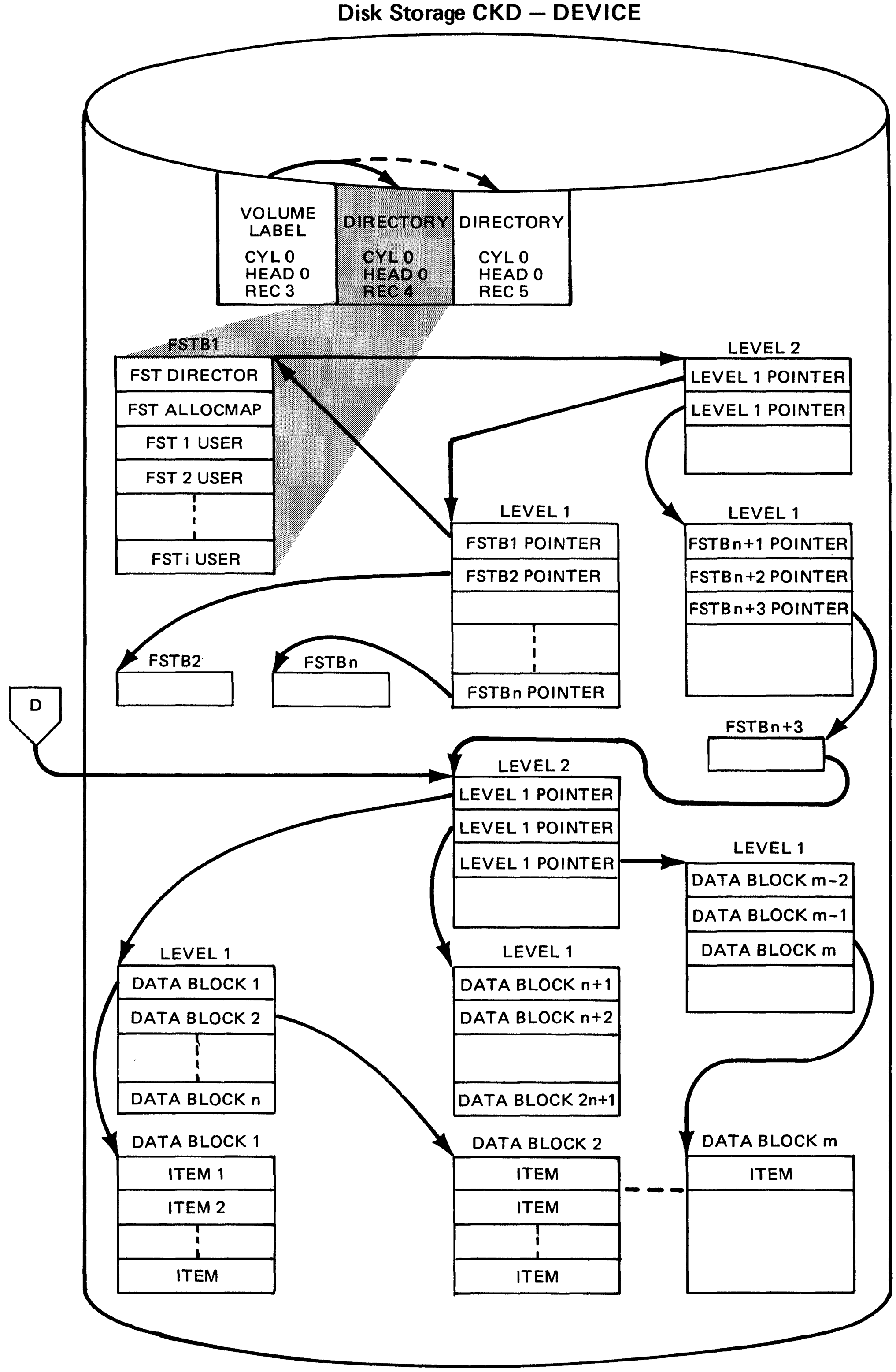


Figure 24 (Part 3 of 3). File System for 512-, 1K-, 2K-, or 4K-Byte Record on Disk

CMS Routines Used to Access the File System

DMSACC is the control routine used to access a virtual disk. In conjunction with DMSACM and DMSACF, DMSACC, DMSACP, and DMSACS build, in virtual storage, the tables CMS requires for processing files contained on the disk. The list below shows the logical flow of the main function of DMSACC.

Access a Virtual Disk: DMSACC

DMSACC
Scans the command line to determine which disk is specified.

DMSLAD
Looks up the address of the ADT for the disk specified on the command line.

DMSACC
Determines whether an extension to a disk has been specified on the command line, and ensures that it is correctly specified.

DMSLAD
In the case where an extension has been specified, calls DMSLAD to ensure that the extension disk exists.

DMSLAD
Ensures that the specified disk is not already accessed as a R/W disk.

DMSFNS
In the case where the specified disk is replacing a currently accessed disk, closes any open files belonging to the duplicate disk.

DMSACC
Verifies the parameters remaining on the command line.

DMSACP
Sets up control blocks for the remaining access processing.

DMSALU
Releases any free storage belonging to the duplicate disk via a call to DMSFRE. Also, clears appropriate entries in the ADT for use by the new disk.

DMSACM
(Called as the first instruction by DMSACF) Reads from the file directory and the allocation map for the specified disk. Also, DMSACM updates the ADT for the specified disk using information from the file directory and disk label.

DMSACF
Reads into storage all the FST blocks associated with the specified disk. DMSACF calls DMSHTB to build hyperblock mapping tables for read/only disks (if the hyperblocks that are searched span three or more pages). DMSACF also calls DMSHTB to build hash tables for

read/write disks (if the hyperblocks that are searched span two or more pages).

DMSACS
Called by DMSACF to load a DCSS containing the FST blocks and hyperblock mapping tables for the disk if shared storage access is allowed for the disk.

DMSACG
If sufficient storage is available and if the disk in question is EDF but not an S- or Y-disk, DMSACG is called by DMSACF. The directory is read into contiguous storage by hyperblocks and sorted, if necessary, bypassing the call to DMSALU (SORTFST). Control then returns to DMSACF.

DMSACP
Handles error processing or processing required to return control to DMSINT.

The SET HASH Command

The SET HASH OFF command disables fileid hashing.

SET	**HASH OFF**

Programs that modify the fileid fields in a file status table (FST) when writing files to a disk can cause a "file not found" condition for the file until the disk is re-accessed. Disabling fileid hashing can alleviate this problem if encountered. Programs that modify any of the fields in the FSTs are not supported by CMS.

Fileid hashing reduces the paging overhead when searching for files on R/W disks. Disabling fileid hashing may degrade performance for the CMS user.

Once disabled, fileid hashing can only be re-enabled by re-IPLing CMS.

Use the QUERY HASH command to determine whether fileid hashing is enabled (set on) or disabled (set off).

Chapter 10. Handling I/O Operations

CMS input/output operations for unit record, disk, and tape devices are always synchronous.

Input/output operations to a card reader, card punch, or printer are initiated via a normal START I/O instruction. After starting the operation, CMS enters the wait state until a device end interruption is received from the started device. Because the I/O is spooled by CP, CMS does not handle any exceptional conditions other than not ready, end-of-file, or forms overflow.

Disk and tape I/O is initiated via a privileged instruction, DIAGNOSE, whose function code requests CP to perform necessary error recovery. Control is not returned to CMS until the operation is complete, except for tape rewind or rewind and unload operations, which return control immediately after the operation is started. No interruption is ever received as the result of DIAGNOSE I/O. The CSW is stored only in the event of an error.

CMS input/output operations to the terminal may be either synchronous or asynchronous. Output to the terminal is always asynchronous, but a program may wait for all terminal input/output operations to complete by calling the console wait routine. Input from the terminal is usually synchronous but a user may cause CMS to issue a read by pressing the attention key. A program may also asynchronously stack data to be read by calling the console attention routine.

Unit Record I/O Processing

Seven routines handle I/O processing for CMS: DMSRDC, DMSPUN, and DMSPRT handle the READCARD, PUNCH, and PRINT commands and pass control to the actual I/O processors, DMSCIO (for READCARD and PUNCH) or DMSPIO (for PRINT). DMSCIO and DMSPIO issue the SIO instructions that cause I/O to take place. Two other routines, DMSIOW and DMSITI, handle synchronization processing for I/O operations. Figure 25 on page 142 shows the overall flow of control for I/O operations.

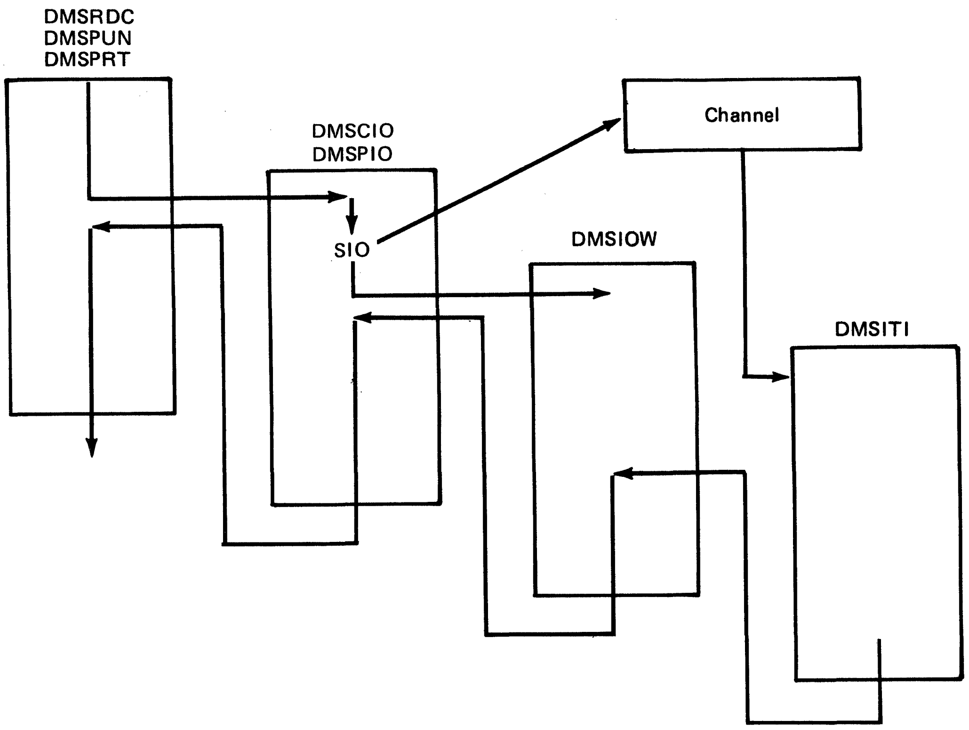


Figure 25. Flow of Control for Unit Record I/O Processing

The following are more detailed descriptions of the flow of control for the read, punch, and print unit record control functions.

Read a Card

DMSRDC
Initializes block length and unit record size.

DMSCIO
Initializes areas to read records.

DMSCIO
Issues an SIO command to read a record.

DMSIOW
Sets the wait bit for the virtual card reader, and loads the I/O old PSW from NUCON. This causes CMS to enter a wait state until the read I/O is complete.

DMSITI
Ensures that this interrupt is for the virtual reader. If not, the I/O old PSW is loaded, returning CMS to a wait state. If the interrupt is for the reader, DMSITI resets the wait bit in the I/O old PSW and loads it causing control to return to DMSIOW.

DMSIOW
Places the symbolic name of the interrupting device in the PLIST, and passes control to the calling routine.

DMSCIO
Checks for SENSE information, and handles I/O errors, if necessary.

DMSCWR
Displays a control record at the console.

DMSSCN
If another control record is encountered, formats it via DMSSCN.

DMSCWR
Displays the new control record at the console.

DMSFNS
Closes the file when end-of-file occurs.

DMSRDC
Issues a CP CLOSE command to close the card reader.

Punch a Card

DMSPUN
Ensures that a virtual punch is available, and processes PUNCH command options.

DMSSTT
Verifies the existence of the file, and returns its starting address.

DMSPUN
If requested, sets up a header record, and calls DMSCWR to write it to the console.

DMSBRD
Reads a block of data into the read buffer, and continues reading until the buffer is filled.

DMSBWR
Writes a block of data on disk.

DMSCIO
Initializes areas to punch records.

DMSCIO
Issues the SIO instruction to punch the contents of the buffer.

DMSCIO
Issues a call to DMSIOW to wait for completion of the punch I/O operation.

DMSIOW
Sets the wait bit on for the virtual punch device, and loads the I/O old PSW from NUCON. This causes CMS to enter a wait state until the punch operation completes.

DMSITI
Ensures that this interrupt is for the punch. If not, the I/O old PSW is loaded returning CMS to a wait state. If the interrupt is for the punch, DMSITI resets the wait bit in the I/O old PSW and then loads the PSW, returning control to DMSIOW.

DMSIOW
Places the symbolic name of the interrupting device in the PLIST, and passes control to DMSCIO.

DMSCIO
Checks for SENSE information, and handles I/O errors, if any.

DMSPUN
Handles error returns, and resets constants for the next punch operation.

DMSFNS
Closes the file, and returns control to the command handler, DMSINT.

Print a File

DMSPRT
Determines the device type of the printer. Checks out the specified fileid. Checks out the options specified on the PRINT command line.

DMSSTT
Verifies the existence of the file, and returns its starting address.

DMSPRT
Determines the record size to be printed, and sets up an appropriate buffer area via a call to DMSFRE.

DMSFRE
Obtains storage space to be used as a buffer.

DMSPRT
Determines whether the file to be printed is a library member or an input file.

DMSBRD
Reads a record; continues reading until the buffer is filled. When the buffer is filled, calls DMSPIO to issue the SIO instruction to begin the print operation.

DMSPIO
Builds appropriate printer CCW chain. Issues the print SIO instruction, and then calls DMSIOW to wait until the I/O operation completes.

DMSIOW
Sets the wait bit for the virtual printer device, and loads the I/O old PSW from NUCON. This causes CMS to enter a wait state until the print operation completes.

DMSITI
Ensures that the interrupt is for the printer. If not, the I/O old PSW is reloaded, returning CMS to a wait state. If the interrupt is for the printer, DMSITI resets the WAIT bit in the I/O old PSW and loads that PSW, returning control to DMSIOW.

DMSIOW
Places the symbolic name of the device in the last word of the PLIST, and passes control to DMSPIO.

DMSPIO
Performs channel testing and handles errors. TIO instructions and sense SIO instructions are issued during the test processing. These operations are synchronized using DMSIOW and DMSITI in the manner described above. When the I/O completes successfully, control returns to DMSPRT.

DMSPRT
Determines whether all file records have been printed. If so, control returns to the caller. Otherwise, the address of the buffer is updated and more print operations are performed.

Printer Carriage Control Characters Used by DMSPIO

CMS supports the use of ASA control characters and machine carriage control characters for the printed output. Part of the CMS implementation depends upon the fact that the set of ASA control characters has almost nothing in common with the set of machine control characters. There are two exceptions to this, the characters X'C1' and X'C3'.

These two characters, when interpreted as ASA control characters, have the following meanings:

C1 = Skip to channel 10 before print.
C3 = Skip to channel 12 before print.

The same characters, when interpreted as machine control characters, have the following meanings:

C1 = Write, then skip to channel 8 after print.
C3 = Do not write, but skip to channel 8 immediately.

In printed lines containing carriage control characters, CMS can operate in two modes. In the first mode, ASA control characters or machines control characters are recognized and properly interpreted. However, two conflicting characters are always interpreted as ASA control characters. In the second mode, only machine control characters are recognized. Two conflicting characters are treated as machine control characters.

The DMSPIO function uses a bit in the PLIST to indicate which of the two modes is in effect for printing.

The PRINTL macro always uses ASA control character mode or machine control character mode.

The PRINT command with the CC option always runs in ASA control character mode or machine control character mode.

OS simulation output, which is used, for example, by the MOVEFILE command, uses the RECFM field in the DCB or in the FILEDEF command to determine which mode is to be used. If FA, VA, or UA is specified, then ASA control character mode or machine control character mode is used. If FM, VM, or UM is specified, then machine-only mode is used. If no control character specification is included with the RECFM, then it is assumed that the output line begins with a valid data character rather than with a control character, and single spacing is always used.

The SETPRT Command

The CMS SETPRT command allows a CMS user to control the facilities of a virtual 3800 device defined for their virtual machine. The SETPRT command is similar in function to the OS SETPRT macro. It allows the user to request multiple character arrangement tables, loading of copy modification, etc. The command uses the current CMS search order for locating disk files. Therefore, users can create their own character arrangement table, copy modifications, etc. and print files with user-defined characteristics. The SETPRT command writes 3800 CCWs and data to a virtual 3800 spool file to set up the real 3800 for the data to follow. If a file is created on a virtual 3800 and printed on a real printer of a different type, the 3800 load CCWs imbedded within the file are ignored and printing takes place as normal. However, this may create output that does not appear as originally intended.

The format of the command is:

SETPRT	[**CHARS** [(] *cccc* ...[)]] [**COPIES** [(] *nnn* [)]] [**COPYNR** [(] *nnn* [)]] [**FCB** [(] *ffff* [)]] [**FLASH** [(] *id nn* [)]] [**INIT**] [**MODIFY** [(] *mmmm* [*n*] [)]]

Note: Due the change in pel density, customized 3800 Model 1 character sets are not interchangeable with the 3800 Model 3 character sets. Users can recode customized 3800 Model 1 character sets and build new modules through the use of the GENIMAGE command. The MVS Character Conversion Aid may also be used to convert existing customized character sets to the 3800 Model 3 pel density.

DMSSPR process the SETPRT command in the following manner:

1. Accept input PLIST and analyze. If there are errors, issue a message to the user and exit.

2. Select the correct character set modules, and load these modules into free storage.

3. Assign writeable character generation modules (WCGMs), and change the translate tables if necessary.

4. Issue SIOs to the virtual 3800 printer. In the case of an error, terminate processing, and issue a message and appropriate return code.

5. Exit with a zero return code if the operation completes successfully.

Disk I/O in CMS

Files residing on disk are read and written using DMSDIO. DMSDIO has two entry points: DMSDIOR, which is entered for a read I/O operation, and DMSDIOW, which is entered for a write operation.

The actual disk I/O operation is performed using the DIAGNOSE code X'18' instruction. A return code of 0 from CP indicates a successful completion of the I/O operation. If the I/O is not successful, CP performs error recording, retry, recovery, or ABEND procedures for the virtual machine.

Read or Write Disk I/O

DMSDIO
Initialize the CCW to perform read operations.

DMSLAD
Obtain the address of the disk from which to read or write.

DMSDIO
Determines the size of the record to be read or written.

DMSFRE
Gets enough storage to contain the record if the request is for a record longer than 800 bytes.

DMSDIO
Reads records continually until all records for the file have been read.

DMSFRE
Returns the buffer to free storage if the record was longer than 800 bytes.

DMSDIO
Returns to the caller.

CMS Tape Label Processing

DMSLBD
Allows the user to specify tape label information that will be used by a program at execution time.

DMSTLB
Processes IBM standard tape labels for OS simulation, CMS/DOS, CMS commands, and the TAPESL macro. It also provides linkage to nonstandard user label routines for OS simulation and CMS commands. There are common tape label checking routines for input header and trailer labels and common tape label writing routines for output header and trailer labels. These common routines are used for all IBM standard label processing regardless of what operating system is being simulated. For OS simulation, DMSTLB also provides multivolume tape processing and merges the HDR2 information into the FCB.

DMSTIO
Reads or writes a tape record. Also performs tape control operations. Functions by issuing DIAGNOSE code X'20'.

DMSTVS
Provides tape volume switching for OS simulation.

Chapter 11. Handling Interruptions

Part 3 of Figure 8 on page 59 lists the CMS modules that process interruptions for CMS. These CMS modules are described briefly in Chapter 15, "Module Entry Point Directory" on page 243. Also, see Chapter 2, "Interrupt Handling in CMS" on page 11.

Chapter 12. Managing CMS Storage

You can allocate free storage by issuing the GETMAIN or DMSFREE macros.

Storage allocated by the GETMAIN macro is taken from the user program area, starting after the high address of the user program. Storage allocated by the DMSFREE macro can be taken from several areas. First, DMSFREE requests are allocated from the low-address free storage area. Otherwise, DMSFREE requests are satisfied from the unused portion of the user program area.

There are two types of DMSFREE requests for free storage: requests for USER storage and NUCLEUS storage, specified in the TYPE parameter of the DMSFREE macro. These two types of storage are kept in 4K pages. It is possible, if there are no 4K pages completely free in low storage, for storage of one type to be available in low storage, while no storage of the other type is available.

GETMAIN Free Storage Management

All GETMAIN storage is allocated in the user program area, starting after the end of the user's actual program. Allocation begins at the location pointed to by the NUCON pointer MAINSTRT. The location MAINHIGH in NUCON points to the highest address of GETMAIN storage.

The STRINIT Macro

The STRINIT function initializes pointers used by CMS for simulation of OS GETMAIN/FREEMAIN storage management. In the usual CMS execution environment, that is, when execution is initiated by the LOAD and START commands, CMS calls the STRINIT function as a part of the LOAD preparation for execution. In an OS environment established by CMS, such as OSRUN, CMS executes the STRINIT function. This should not be done by the user program. In any case, the STRINIT macro should be issued only once in the OS environment preceding the initial GETMAIN request. Also, the STRINIT function will make any pages allocated by GETMAIN available to be released by the CMS page manager.

The format of the STRINIT macro is:

[*label*]	**STRINIT**	$\left[\textbf{TYPCALL}=\left\{\begin{matrix}\underline{\textbf{SVC}}\\ \textbf{BALR}\end{matrix}\right\}\right]$

where:

$\textbf{TYPCALL}=\left\{\begin{matrix}\underline{\textbf{SVC}}\\ \textbf{BALR}\end{matrix}\right\}$

indicates how control is passed to DMSSTG, the routine that processes the STRINIT macro. Since DMSSTG is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL=BALR). Routines that are not nucleus-resident must use SVC linkage (TYPCALL=SVC). If no operands are specified, the default is TYPCALL=SVC.

When the STRINIT macro is executed, both MAINSTRT and MAINHIGH are initialized to the end of the user's program in the user program area. The end of the user's program is the upper boundary of the load module created by the CMS LOAD and INCLUDE commands. This upper boundary value is stored in the NUCON field LOCCNT. When the user's program begins execution, the STRINIT macro is executed and the LOCCNT value is used to initialize MAINSTRT and MAINHIGH. During execution of the user's program, the LOCCNT field is used in CMS to pass starting and ending addresses of files loaded by OS simulation. (Reissuing the STRINIT macro during execution of an OS program, or issuing the STRINIT macro without having done a CMS LOAD is not advised. The value in LOCCNT has not been appropriately set and this may cause a storage management failure.) As storage is allocated from the user program area to satisfy GETMAIN requests, the MAINHIGH pointer is adjusted upward. Such adjustments are always in multiples of doublewords, so that this pointer is always on a doubleword boundary. As the allocated storage is returned, the MAINHIGH pointer is adjusted downward. When the user issues a variable length GETMAIN, the control program reserves 6.5 pages for CMS usage; this is a designed and set value. If the user wants more space, (for example, for more directories) the user should free some of the variable GETMAIN area from the high end.

The pointer MAINHIGH can never be higher than FREELOWE. FREELOWE is the pointer to the lowest address of DMSFREE storage allocated in the user program area. If a GETMAIN request cannot be satisfied without extending MAINHIGH above FREELOWE, GETMAIN takes an error exit, indicating that insufficient storage is available to satisfy the request.

The area between MAINSTRT and MAINHIGH may contain blocks of storage that are not allocated, but these blocks are available for allocation by a GETMAIN instruction. These blocks are chained together, and the first block is pointed to by the NUCON location MAINLIST. See Figure 3 on page 20 for a description of CMS virtual storage usage.

The format of an element on the GETMAIN free element chain is as follows:

0(0)	FREPTR – pointer to next free element in the chain, or 0 if there is no next element
4(4)	FRELEN – length, in bytes, of this element
	Remainder of this free element . . .

The maximum amount of storage that can be obtained via the GETMAIN macro is determined by one of the following formulas:

VMSIZE < 512K:

(largest block of the user program area available) - 10 pages

VMSIZE > = 512K:

(largest block of the user program area available) - (12 pages + 2 additional pages for each 256K of virtual storage over 512K)

Releasing Storage

Storage allocated by the GETMAIN macro instruction may be released in any of the following ways:

- A specific block of such storage may be released by means of the FREEMAIN macro instruction.
- Whenever any user routine or CMS command abends (so that the routine DMSABN is entered) and the ABEND recovery facility of the system is invoked, all GETMAIN storage area pointers are reset.
- Issuing a STRINIT macro releases all allocated GETMAIN storage.

DMSFRE Free Storage Management

The DMSFREE Macro

The DMSFREE macro allocates CMS free storage. The format of the DMSFREE macro is:

$\left[\,\textit{label}\,\right]$	**DMSFREE**	**DWORDS=** $\begin{Bmatrix} n \\ \textbf{(0)} \end{Bmatrix}$ $\left[\textbf{,MIN=}\begin{Bmatrix} n \\ \textbf{(1)} \end{Bmatrix}\right]$ $\left[\textbf{,TYPE=}\begin{Bmatrix} \underline{\textbf{USER}} \\ \textbf{NUCLEUS} \end{Bmatrix}\right]$ $\left[\textbf{,ERR=}\begin{Bmatrix} \textit{laddr} \\ * \end{Bmatrix}\right]$ $\left[\textbf{,AREA=}\begin{Bmatrix} \textbf{LOW} \\ \textbf{HIGH} \end{Bmatrix}\right]$ $\left[\textbf{,TYPCALL=}\begin{Bmatrix} \underline{\textbf{SVC}} \\ \textbf{BALR} \end{Bmatrix}\right]$

where:

label
: is any valid assembler language label.

DWORDS= $\begin{Bmatrix} n \\ \textbf{(0)} \end{Bmatrix}$
: is the number of doublewords of free storage requested. DWORDS=*n* specifies the number of doublewords directly. DWORDS=(0) indicates that register 0 contains the number of doublewords requested.

Do not specify any register other than register 0. The register number for register 0 cannot be expressed as an equated symbol. CMS returns, in register 0, the number of doublewords allocated and, in register 1, the address of the first byte of allocated storage.

MIN= $\begin{Bmatrix} n \\ \textbf{(1)} \end{Bmatrix}$
: indicates a variable request for free storage. If the exact number of doublewords indicated by the DWORDS operand is not available, the largest block of storage greater than or equal to the minimum is returned. MIN=*n* specifies the minimum number of doublewords of free storage directly. MIN=(1) indicates that the minimum is in register 1. Do not specify any register other than register 1. The actual amount of free storage allocated is returned to the requestor via general register 0.

TYPE = $\left\{\begin{array}{l}\underline{\textbf{USER}}\\ \textbf{NUCLEUS}\end{array}\right\}$

indicates the type of CMS storage requested: USER or NUCLEUS.

ERR = $\left\{\begin{array}{l}\textit{laddr}\\ *\end{array}\right\}$

is the return address if any error occurs. *laddr* is any address that can be referred to in an LA (load address) instruction. The error return is taken if there is a macro coding error or if there is not enough free storage available to fill the request. If the asterisk (*) is specified for the return address, the error return is the same as a normal return. There is no default for this operand. If it is omitted and an error occurs, the system abends.

AREA = $\left\{\begin{array}{l}\textbf{LOW}\\ \textbf{HIGH}\end{array}\right\}$

indicates the area of CMS free storage requested. LOW indicates any free storage below the user areas, depending on the storage requested. HIGH indicates DMSFREE storage above the user area. If AREA is not specified, storage is allocated whenever it is available.

TYPCALL = $\left\{\begin{array}{l}\underline{\textbf{SVC}}\\ \textbf{BALR}\end{array}\right\}$

indicates how control is passed to DMSFREE. Since DMSFREE is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL = BALR). Routines that are not nucleus-resident must use SVC linkage (TYPCALL = SVC).

The FREELOWE pointer in NUCON indicate the amount of storage that DMSFREE has allocated from the high portion of the user program area. These pointers are initialized to the beginning of the system loader tables.

The pointer FREELOWE is the pointer to the lowest address of DMSFREE storage in the user program area. As storage is allocated from the user program area to satisfy DMSFREE requests, the pointer FREELOWE is adjusted downward. As the allocated storage is returned, this pointer is adjusted upward. Such adjustments are in multiples of 4K bytes so the pointer is always on a 4K boundary. As the allocated storage is returned, this pointer is adjusted upward when whole 4K pages are completely free.

The pointer FREELOWE can never be lower than MAINHIGH. MAINHIGH is the pointer to the highest address of GETMAIN storage. If a DMSFREE request cannot be satisfied without extending FREELOWE below MAINHIGH, DMSFREE takes an error exit, indicating that insufficient storage is available to satisfy the request. Figure 3 on page 20 shows the relationship of these storage areas.

The FREETAB free storage table is usually kept in nucleus low free storage. However, the FREETAB may be located at the top of the user program area. This table contains a code indicating the use of that page of virtual storage.

The codes in this table are as follows:

Code	**Meaning**
USERCODE (X'01')	The page is assigned to user storage.
NUCCODE (X'02')	The page is assigned to nucleus storage.
TRNCODE (X'03')	The page is part of the transient program area.
USARCODE (X'04')	The page is an unassigned page in the user program area.
SYSCODE (X'05')	The page is none of the above. The page is assigned to system storage, system code, or the loader tables.

Other DMSFREE storage pointers are maintained in the DMSFRT CSECT, in NUCON. The four chain header blocks are the most important fields in DMSFRT. The four chains of unallocated elements are:

- The low storage nucleus chain
- The low storage user chain
- The high storage nucleus chain
- The high storage user chain.

For each of these chains of unallocated elements, there is a control block consisting of four words with the following format:

<table>
<tr><td>0(0)</td><td colspan="4">POINTER – pointer to the first FBD (free block descriptor) in a cache of FBDs used to describe the first "n" free blocks of storage for the particular chain.</td></tr>
<tr><td>4(4)</td><td colspan="4">NUM – the number of elements on the chain</td></tr>
<tr><td>8(8)</td><td colspan="4">MAX – a value equal to or greater than the size of the largest free element on the chain</td></tr>
<tr><td>12(C)</td><td>FLAGS – Flag byte</td><td>SKEY – Storage key</td><td>TCODE – FREETAB code</td><td>Unused</td></tr>
</table>

where:

POINTER

points to the first FBD (free block descriptor) in a cache of FBDs used to describe the first "n" free blocks of storage for the particular chain. "n" is 10 for the high user chain, 9 for the high nucleus chain, 6 for the low user chain, and 6 for the low nucleus chain.

NUM

contains the number of elements on this chain of free elements. If there are no elements on this free chain, this field contains all zeros.

MAX

is used to avoid searches that will fail. It contains a number not exceeding the size, in bytes, of the largest element on the free chain. Thus, a search for an element of a given size is not made if that size exceeds the MAX field. However, this number may actually be larger than the size of the largest free element on the chain.

FLAGS

The following flags are used:

FLCLN (X'80') - Clean-up flag. This flag is set if the chain must be updated. This is necessary in the following circumstances:

- If one of the two high-storage chains contains a 4K page that is pointed to by FREELOWE, that page can be removed from that chain, and FREELOWE can be increased.

- All completely unallocated 4K pages are kept on the user chain, by convention. Thus, if one of the nucleus chains (low-storage or high-storage) contains a full page, this page must be transferred to the corresponding user chain.

FLCLB (X'40') - Destroyed flag. Set if the chain has been destroyed.

FLHC (X'20') - High-storage chain. Set for both the nucleus and user high-storage chains.

FLNU (X'10') - Nucleus chain. Set for both the low-storage and high-storage nucleus chains.

FLPA (X'08') - Page available. Set if there is a full 4K page available on the chain. This flag may be set even if there is no such page available.

SKEY

contains the one-byte storage key assigned to storage on this chain.

TCODE

contains the one-byte field FREETAB table code for storage on this chain.

There are four caches of FBDs, one for each of the chains. The FBDs are chained together at initialization time from the head pointers found in the DMSFRT CSECT described above.

Each of the FBDs in the cache has the following format:

	◄— 4 bytes —►
0(0)	POINTER – pointer to the next FBD in the chain unless it is the last FBD in the cache in which case it points to the next block of free storage in the chain or is zero.
4(4)	SIZE – size of the free block in bytes
8(8)	FBDFREE – pointer to the free block that this FBD is describing.

The FBDs in the cache always remain chained together, and when they do not describe a free block, the fields SIZE and FBDFREE are zero. When the cache is full, the forward pointer POINTER in the last FBD in the cache points to the next free block that contains the following fields:

	◄— 4 bytes —►
0(0)	POINTER – pointer to the next element in the free chain or is zero
4(4)	SIZE – size of this free element, in bytes
	Remainder of this free element . . .

As indicated in the illustration above, the POINTER field points to the next element in the chain, or contains the value zero if there is no next element. The SIZE field contains the size of this element, in bytes.

The eight bytes before the first physical FBD in each cache contains eight bytes of information about the cache and has the following fields:

	◄— 4 bytes —►
0(0)	CHILAST – last FBD in the cache of free pointers. The forward pointer in this FBD points to the first POINTER off the cache or is zero if there are none.
4(4)	CHINUM – the number of FBDs in the cache
8(8)	CHIFLAG – a flag field used by storage management.

All elements within a given chain are chained together in order of descending storage address. This is done for two reasons:

1. Because the allocation search is satisfied by the first free element that is large enough, the allocated elements are grouped together at the top of the storage area, and prevent storage fragmentation. This is

particularly important for high-storage free storage allocations, because it is desirable to keep FREELOWE as high as possible.

2. If free storage does become somewhat fragmented, the search causes as few page faults as possible.

As a matter of convention, completely nonallocated 4K pages in high storage are kept on the user free chain rather than the nucleus free chain. This is because requests for large blocks of storage are made, most of the time, from user storage rather than from nucleus storage. Nucleus requests need to break up a full page less frequently than user requests.

Allocating User Free Storage

When DMSFREE with TYPE=USER (the default) is called, the following steps are taken to satisfy the request. As soon as one of the following steps succeeds, user free storage allocation processing terminates and the CMS page manager is notified of any full or partial pages that have been allocated.

1. Search the low-storage user chain for a block of the required size.

2. Search the high-storage user chain for a block of the required size.

3. Extend high-storage user storage downward into the user program area, modifying FREELOWE in the process.

4. For fixed requests, there is nothing more to try. For variable requests, DMSFREE puts all available storage in the user program area onto the high-storage user chain, and then allocates the largest block available on either the high-storage user chain or the low-storage user chain. The allocated block is not satisfactory unless it is larger than the minimum requests size.

Allocating Nucleus Free Storage

When DMSFREE with TYPE=NUCLEUS is called, the following steps are taken to satisfy the request. As soon as one of the following steps succeeds, user free storage allocation processing terminates and the CMS page manager is notified of any full or partial pages that have been allocated.

1. Search the low-storage nucleus chain for a block of the required size.

2. Search the high-storage nucleus chain for a block of the required size.

3. Get free pages from the high-storage user chain, if they are available, and put them on the high-storage nucleus chain.

4. Extend high-storage nucleus storage downward into the user program area, modifying FREELOWE in the process.

5. For fixed requests, there is nothing more to try. For variable requests, DMSFRE puts all available pages from the high-storage user chain and

the user program area onto the high-storage nucleus chain, and allocates the largest block available on either the low-storage nucleus chain or the high-storage nucleus chain.

Releasing Storage

Storage allocated by the DMSFREE macro instruction may be released in either of the following ways:

- A specific block of such storage may be released by means of the DMSFRET macro instruction.
- Whenever any user routine on CMS command abnormally terminates (so that the routine DMSABN is entered) and the abend recovery facility of the system is invoked, all DMSFREE storage with TYPE=USER is released automatically.

Except in the case of abend recovery, storage allocated by the DMSFREE macro is never released automatically by the system. Thus, storage allocated by means of this macro instruction should always be released explicitly by means of the DMSFRET macro instruction. Whenever a completely unused 4K page becomes available, it is made eligible for release by a call to the CMS page manager.

The DMSFRET Macro

The DMSFRET macro releases free storage previously allocated with the DMSFREE macro.

The format of the DMSFRET macro is:

$\left[\,\mathit{label}\,\right]$	**DMSFRET**	**DWORDS=** $\begin{Bmatrix} \mathit{n} \\ \mathbf{(0)} \end{Bmatrix}$ **,LOC =** $\begin{Bmatrix} \mathit{laddr} \\ \mathbf{(1)} \end{Bmatrix}$ $\left[\mathbf{,ERR} = \begin{Bmatrix} \mathit{laddr} \\ \mathbf{*} \end{Bmatrix}\right]$ $\left[\mathbf{,TYPCALL} = \begin{Bmatrix} \underline{\mathbf{SVC}} \\ \mathbf{BALR} \end{Bmatrix}\right]$

where:

label

is any valid Assembler language label.

DWORDS= $\begin{Bmatrix} \mathit{n} \\ \mathbf{(0)} \end{Bmatrix}$

is the number of doublewords of storage to be released. DWORDS=*n* specifies the number of doublewords directly. DWORDS=(0) indicates that register 0 contains the number of doublewords being released. Do not specify any register other than register 0. The register number for register 0 cannot be expressed as an equated symbol.

LOC = $\left\{\begin{array}{l}\textit{laddr}\\ \textbf{(1)}\end{array}\right\}$

is the address of the block of storage being released. *laddr* is any address that can be referred to in an LA (load address) instruction. LOC = *laddr* specifies the address directly. LOC = (1) indicates the address is in register 1. Do not specify any register other than register 1.

ERR = $\left\{\begin{array}{l}\textit{laddr}\\ \textbf{*}\end{array}\right\}$

is the return address if an error occurs. *laddr* is any address that can be referred to by an LA (load address) instruction. The error return is taken if there is a macro coding error or if there is a problem returning the storage. If an asterisk (*) is specified, the error return address is the same as the normal return address. There is no default for this operand. If it is omitted and an error occurs, the system abend.

TYPCALL = $\left\{\begin{array}{l}\underline{\textbf{SVC}}\\ \textbf{BALR}\end{array}\right\}$

indicates how control is passed to DMSFRET. Since DMSFRET is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL = BALR). Routines that are not nucleus-resident must use SVC linkage (TYPCALL = SVC).

When DMSFRET is called, the block being released is placed on the appropriate chain. At that point, the final update operation is performed, if necessary, to advance FREELOWE, or to move pages from the nucleus chain to the corresponding user chain.

Similar update operations are performed, when necessary, after calls to DMSFREE, as well. The CMS page manager is notified of any completely unallocated page.

CMS Page Management

The CMS page manager (DMSPAG) controls the release of allocated storage. When the CMS page manager is notified of a completely nonallocated 4K page, that page is made available for release. The page manager holds the available pages, and when the number exceeds a system-defined maximum, those pages are released via DIAGNOSE code X'10'. If storage management routines allocate any part of a 4K page being held, that page is no longer available for release.

You can stop the release of available pages by issuing the SET RELPAGE OFF command. The page manager continues to track pages, and when you set RELPAGE ON, all available pages are released.

DMSFRE Service Routines

The system uses the DMSFRES macro to request certain free storage management services.

The DMSFRES Macro

The format of the DMSFRES macro is:

[*label*]	**DMSFRES**	**INIT1** **INIT2** **CHECK** **CKON** **CKOFF** **UREC** **CALOC**	[**,TYPCALL** = { **SVC** **BALR** }]

where:

label
is any valid Assembler language label.

INIT1
invokes the first free storage initialization routines to allow free storage requests to access the system disk. Before INIT1 is invoked, no free storage requests may be made. After INIT1 has been invoked, free storage requests may be made. However, these requests are subject to the following restraints until the second free storage management initialization routine has been invoked:

- All requests for USER storage are changed to requests for NUCLEUS storage.

- Error checking is limited before initialization is complete. In particular, it is sometimes possible to release a block that was never allocated.

- All requests that are satisfied in high storage must be temporary, because all storage allocated in high storage is released when the second free storage initialization routine is invoked.

When CP's saved system facility is used, the CMS system is saved at the point after the system disk has been accessed. It is necessary for DMSFRE to be used before the size of virtual storage is known, because the saved system can be used on any size virtual machine. Thus, the first initialization routine initializes DMSFRE so that limited functions can be requested. The second initialization routine performs the initialization necessary to allow the full functions of DMSFRE to be exercised.

INIT2

invokes the second initialization routine. This routine is invoked after the size of virtual storage is known, and it performs initialization necessary to allow all the functions of DMSFRE to be used. The second initialization routine performs the following steps:

- Releases all storage that has been allocated in the high-storage area.

- Allocates the FREETAB free storage table. This table contains one byte for each 4K page of virtual storage. Therefore, the table cannot be allocated until the size of virtual storage is known. It is allocated in the nucleus low free storage area, if there is enough room available. If not, then it is allocated in the higher free storage area. For a 256K virtual machine, FREETAB contains 64 bytes; for a 16 million byte machine, it contains 4096 bytes.

- The FREETAB and all storage protection keys are initialized.

CHECK

invokes a routine that checks all free storage pointer chains for consistency and correctness. Thus, it checks to see whether any free storage pointers have been destroyed. This option can be used at any time for system debugging.

CKON

turns on a flag that causes the CHECK routine to be invoked each time a call is made to DMSFREE or DMSFRET. This can be useful for debugging purposes (for example, when you wish to identify the routine that destroyed free storage management pointers). Care should be taken when using this option, since the CHECK routine is coded to be thorough rather than efficient. Thus, after the CKON option has been invoked, each call to DMSFREE or DMSFRET takes much longer to be completed than before. This can impact the efficiency of system functions.

CKOFF

turns off the flag that was turned on by the CKON option.

UREC

is used by DMSABN during the abend recovery process to release all user storage.

CALOC

is used by DMSABN after the abend recovery process has been completed. It invokes a routine that returns, in register 0, the number of doublewords of free storage that have been allocated. This number is used by DMSABN to determine whether the abend recovery has been successful.

TYPCALL = { SVC | BALR }

indicates how control is passed to DMSFRES. Since DMSFRES is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL=BALR). Routines that are not nucleus-resident must use SVC linkage (TYPCALL=SVC).

Storage Protection Keys

In general, the following rule for storage protection keys applies: system storage is assigned the storage key of X'F0', while user storage is assigned the storage key of X'E0'. This is the storage key associated with the protected areas of storage, not to be confused with the PSW or CAW key used to access that storage.

The specific key assignments are as follows:

- The NUCON area is assigned the key of X'F0', with the exception of the last page containing the OPSECT and TSOBLOKS areas and user free storage, which have a key of X'E0'.
- Free storage allocated by DMSFREE is broken up into user storage and nucleus storage. The user storage has a protection key of X'E0', while the nucleus storage has a key of X'F0'.
- The transient program area has a key of X'E0'.
- The CMS nucleus code has a storage key of X'00'. In saved systems, this entire segment is protected by CP from modification even by the CMS system, and so must be entirely reentrant.
- The user program area is assigned the storage key of X'E0', except for those pages which contain nucleus DMSFREE storage. These latter pages are assigned the key of X'F0'.
- The loader tables are assigned the key of X'F0.

The SET KEYPROTECT Command

The SET KEYPROTECT command controls the resetting of user keys, X'E0', when a DMSFRET occurs. The format of the SET KEYPROTECT command is:

SET	**KEYPROTect { ON OFF }**

When you issue SET KEYPROTECT ON, the storage keys for the whole virtual machine, except the nonshared pages, are reset according to FREETAB. Then whenever a DMSFRET occurs, the user keys are reset.

SET KEYPROTECT OFF does not cause the user keys to be reset when a DMSFRET occurs. (SET KEYPROTECT OFF is the default setting.) If an ABEND occurs, the storage keys of the virtual machine are reset according to FREETAB and the setting for KEYPROTECT is maintained.

To check the setting of KEYPROTECT, issue:

```
QUERY KEYPROTECT
```

Note: If user programs set keys, they must restore the keys to their original settings. If there are programs that depend on CMS resetting user keys, SET KEYPROTECT ON to insure that the user keys are set properly.

CMS Handling of PSW Keys

The CMS nucleus protection scheme protects the CMS nucleus from inadvertent destruction by a user program. This mechanism, however, does not prevent you from writing in system storage intentionally. Because you can execute privileged instructions,
you can issue a LOAD PSW (LPSW) instruction and load any PSW key you wish. If this occurs, there is nothing to prevent your program from:

- Modifying nucleus code
- Modifying a table or constant area
- Losing files by modifying a CMS file directory.

In general, user programs and disk-resident CMS commands are executed with a PSW key of X'E', while nucleus code is executed with a PSW key of X'0'.

There are, however, some exceptions to this rule. Certain disk-resident CMS commands run with a PSW key of X'0', because they have a constant need to modify nucleus pointers and storage. The nucleus routines called by the GET, PUT, READ, and WRITE macros run with a user PSW key of X'E' to increase efficiency.

Two macros, DMSKEY and DMSEXS, are available to any routine that wishes to change its PSW key.

The DMSKEY Macro

The DMSKEY macro may be used to change the PSW key to the user value or the nucleus value. The format of the DMSKEY macro is:

<table>
<tr><td>[<i>label</i>]</td><td><b>DMSKEY</b></td><td>{<b>NUCLEUS</b> [<b>,NOSTACK</b>] |
<b>USER</b> [<b>,NOSTACK</b>] |
<b>LASTUSER</b> [<b>,NOSTACK</b>] |
<b>RESET</b>}</td></tr>
</table>

where:

NUCLEUS
causes the nucleus storage protection key to be placed in the PSW, and the old contents of the second byte of the PSW are saved in a stack. This option allows the program to store into system storage, which is ordinarily protected.

USER
causes the user storage protection key to be placed in the PSW, and the old contents of the second byte of the PSW are saved in a stack. This option prevents the program from inadvertently modifying nucleus storage, which is protected.

LASTUSER
The SVC handler traces back through its system save areas for the active user routine closest to the top of the stack. The storage key in effect for that routine is placed in the PSW. The old contents of the second byte of the PSW are saved in a stack. This option should be used only by system routines that should enter a user exit routine. (OS macro simulation routines use this option when they want to enter a user-supplied exit routine. The exit routine is entered with the PSW key of the last user routine on the SVC system save area stack.)

NOSTACK
This option may be used with any of the above options to prevent the system from saving the second byte of the current PSW in a stack. If this is done, then no DMSKEY RESET need be issued later.

RESET
The second byte of the PSW is changed to the value at the top of the DMSKEY stack and removed from the stack. Thus, the effect of the last DMSKEY NUCLEUS, DMSKEY USER, or DMSKEY LASTUSER request is reversed. However, if the NOSTACK option was specified on the DMSKEY macro, the RESET option should not be used. A DMSKEY RESET macro must be executed for each DMSKEY NUCLEUS, DMSKEY USER, or DMSKEY LASTUSER macro that was executed and that did not specify the NOSTACK option. Failure to observe this rule results in program abnormal termination. CMS

requires that the DMSKEY stack must be empty when a routine terminates.

Note: The DMSKEY key stack has a current maximum depth of seven for each routine. In this context, a "routine" is anything invoked by an SVC call.

The DMSEXS Macro

The DMSEXS, "execute in system mode," macro allows a routine executed with a user PSW key to execute a single instruction with a nucleus PSW key. The single instruction may be specified as the argument to the DMSEXS macro, and that instruction is executed with a nucleus PSW key. This macro can be used instead of two DMSKEY macros.

The format of the DMSEXS macro is:

[*label*]	**DMSEXS**	*op-code,operands*

The op-code and the operands of the Basic Assembler Language instruction to be executed must be given as arguments to the DMSEXS macro.

For example, execution of the sequence,

```
USING  NUCON,0
DMSEXS ST,R4,AUSERRST
```

causes the STORE instruction to be executed with a zero protect key in the PSW. This sequence stores the contents of register 4 in the AUSERRST field in the nucleus.

The instruction to be executed may be an EX instruction.

Note: Programs that modify or manipulate bits in CMS control blocks, however, may hinder the operation of CMS causing it to function ineffectively.

Register 1 cannot be used in any way in the instruction being executed.

Whenever possible, CMS commands are executed with a user protect key. This protects the CMS nucleus in cases where there is an error in the system command that would otherwise destroy the nucleus. If the command must execute a single instruction or small group of instructions that modify nucleus storage, then the DMSKEY or DMSEXS macros are used, so that the system PSW key is used for as short a period of time as possible.

CP Handling For Saved Systems

The explanation of saved system nucleus protection depends on the VSK, RSK, VPK and RPK:

1. Virtual Storage Key (VSK) - This is the storage key assigned by the virtual machine using the virtual SSK instruction.

2. Real Storage Key (RSK) - This is the actual storage key assigned by CP to the 2K page.

3. Virtual PSW Key (VPK) - This is the PSW storage key assigned by the virtual machine, by means of an instruction such as LPSW (load PSW).

4. Real PSW Key (RPK) - This is the PSW storage key assigned by CP, which is in the real hardware PSW when the virtual machine is running.

When there are no shared segments in the virtual machine, storage protection works as it does on a real machine. RSK=VSK for all pages, and RPK=VPK for the PSW.

However, when there is a shared segment (as in the case of the CMS nucleus), it is necessary for CP to protect the shared segment. For non-CMS shared segments, CP protects the shared segment by ignoring the values of the VSKs and VPK and assigning the real values as follows: RSK=0 for each page of the shared segment, RSK=F for all other pages, and RPK=F, always, for the real PSW. The SSK instruction is ignored, except to save the key value in a table in case the virtual machine later does an ISK to get back.

For the CMS saved system, the RSKs and RPK are initialized as before, but resetting the virtual keys has the following effects:

- If the virtual machine uses an SSK instruction to reset a VSK, CP does the following: if the new VSK is nonzero, CP resets the RSK to the value of the VSK; if the new VSK is zero, CP resets RSK to F.

- If the virtual machine uses a LPSW (or other) instruction to reset the VPK, CP does the following: If the new VPK is non-zero, CP resets the RPK to the value of the VPK; if the new VPK is zero, CP resets RFK to F.

- If the VPK=0 and the RPK=F, storage protection may be handled differently. In a real machine, a PSW key of 0 would allow the program to store into any storage location, no matter what the storage key. But under CP, the program gets a protection violation, unless the RPK of the page happens to be F.

  Because of this, there is extra code in the CP program check handling program. Whenever a protection violation occurs, CP checks to see if the following conditions hold:

- The virtual machine running is the saved CMS system, running with a shared segment.
- The VPK=0. The virtual machine is operating as though its PSW key is 0.
- The RSK of the page where the store was attempted is nonzero, and different from the RPK.

If any one of these three conditions fails to hold, then the protection violation is reflected back to the virtual machine.

If all three of these conditions hold, then the RPK (the real protection key in the real PSW) is reset to the RSK of the page where the store was attempted.

Effects on CMS

In CMS, this works as follows: CMS keeps its system storage in protect key F (RSK = VSK = F), and user storage in protect key E (RSK = VSK = E).

When the CMS supervisor is running, it runs in PSW key 0 (VPK = 0, RPK = F), so that CMS gets a protection violation the first time it tries to store into user storage (VSK = RSK = E). At that point, CP changes the RPK to E, and lets the virtual machine re-execute the instruction that caused the protection violation. There is not another protection violation until the supervisor goes back to storing into system-protected storage.

Restrictions on CMS

There are several coding restrictions that must be imposed on CMS if it is to run as a saved system.

The first and most obvious one is that CMS may never modify the segments containing CMS nucleus code that is shared and runs with a RSK of 0, although the VSK=F.

A less obvious, but just as important, restriction is that CMS may never modify with a single machine instruction (except MVCL) a section of storage that crosses the boundary between two pages with different storage keys. This restriction applies not only to SS instructions, such as MVC and ZAP, but also to RS instructions, such as STM, and to RX instructions, such as ST and STD, which may have nonaligned addresses on the System/370. An exception is the MVCL instruction. This instruction can be restarted after crossing a page boundary because the registers are updated when the paging exception occurs.

This restriction also applies to I/O instructions. If the key specified in the CCW is zero, then the data area for input may not cross the boundary between two pages with different storage keys.

Overhead

It can be seen that this system is most inefficient when "storage-key thrashing" occurs -- when the virtual machine with a VPK of 0 jumps around, storing into pages with different VSK's.

Error Codes From DMSFRES, DMSFREE, and DMSFRET

For information on nonzero return codes from DMSFRES, DMSFREE, and DMSFRET, see Chapter 17, "DMSFREX Error Codes" on page 279.

Chapter 13. Simulating Non-CMS Environments

The following contains descriptions for: access method support for non-CMS operating systems, CMS simulation of OS functions, and CMS implementation of VSE functions.

OS Access Method Support

An access method governs the manipulation of data. To make the execution of OS generated code easier under CMS, the processing program must see data as OS would present it. For instance, when the processors expect an access method to acquire input source records sequentially, CMS invokes specially written routines that simulate the OS sequential access method and passes data to the processors in the format that the OS access methods would have produced. Therefore, data appears in storage as if it had been manipulated using an OS access method. For example, block descriptor words (BDW), buffer pool management, and variable records are maintained in storage as if an OS access method had processed the data. The actual writing to and reading from the I/O device is handled by CMS file management.

The work of the volume table of contents (VTOC) and the data set control block (DSCB) is done by a master file directory (MFD). The MFD maintains the disk contents and a file status table (FST) for each data file. All disks are formatted in physical blocks of 512, 800, 1024, or 4096 bytes.

CMS continues to maintain the OS format, within its own format, on the auxiliary device for files whose filemode number is 4. That is, the block and record descriptor words (BDW and RDW) are written along with the data. If a data set consists of blocked records, the data is written to and read from the I/O device in physical blocks, rather than logical records. CMS also simulates the specific methods of manipulating data sets.

To accomplish this simulation, CMS supports certain essential macros for the following access methods:

BDAM (direct)
identifying a record by a key or by its relative position within the data set.

BPAM (partitioned)
seeking a named member within an entire data set.

BSAM/QSAM (sequential)
accessing a record in a sequence in relation to preceding or following records.

VSAM (direct or sequential)
accessing a record sequentially or directly by key or address. CMS support of OS VSAM files is based on VSE/VSAM. Therefore, the OS user is restricted to those services available under VSE/VSAM.

CMS Support for the Virtual Storage Access Method

CMS simulation of OS and DOS includes support for the virtual storage access method (VSAM). The description of this support is in three parts:

- A description of the access method services program (AMSERV), which allows you to create and update VSAM files.
- A description of support for VSAM functions under CMS/DOS.
- A description of support for VSAM functions for the CMS OS simulation routines.

  The routines that support VSAM reside in four discontiguous shared segments (DCSSs).

  - The CMSAMS DCSS, which contains the VSE/VSAM code to support AMSERV processing.
  - The CMSVSAM DCSS, which contains actual VSE/VSAM code, and the CMS/VSAM OS interface program for processing OS VSAM requests.
  - The CMSDOS DCSS, which contains the code that supports VSE requests under CMS.
  - The CMSBAM DCSS, which contains the SAM modules required for AMS to access SAM files.

Note: DMSVSR, which performs completion processing for CMS/VSAM support, resides in the CMS nucleus.

Creating the DOSCB Chain

The DLBL command creates a control block called a DOSCB in CMS free storage. The ddname specified in this DLBL command is associated with the ddname parameter in the program's ACB.

The DOSCB contains information defining the file for the system. The information in the DOSCB parallels the information written on the label information area of a real DOS SYSRES unit; for example, the name, and mode (volume serial number) of the data set, its logical unit specification, and its data set type (SAM or VSAM). The anchor for this chain is at location DOSFIRST in NUCON.

Executing an AMSERV Function

The CMS AMSERV command invokes the module DMSAMS, which is the CMS interface to the VSE/VSAM access method services (AMS) program. Module DMSAMS loads VSE/VSAM AMS code, contained in the CMSAMS DCSS, by means of the LOADSYS DIAGNOSE code X'64'. The AMS code requires the services of VSE/VSAM code that resides in the CMSVSAM DCSS. So, that DCSS is also loaded via LOADSYS DIAGNOSE X'64' when the VSAM master catalog is opened. Figure 26 shows the relationship in storage between the interface module DMSAMS, the CMSAMS DCSS, and the CMSVSAM DCSS.

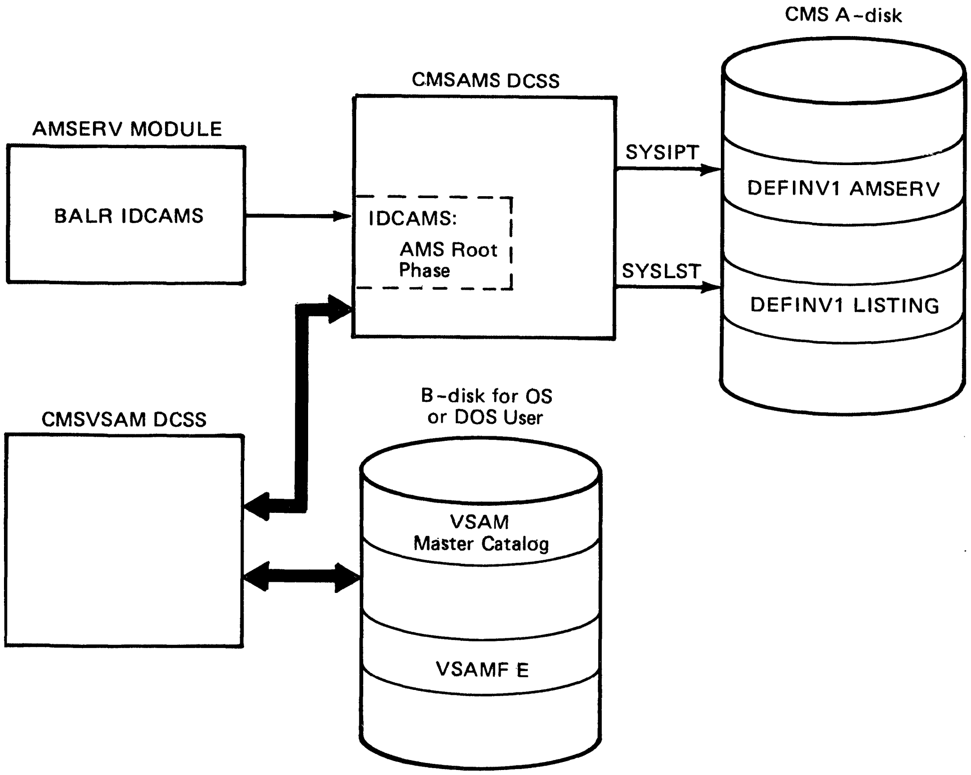


Figure 26. Relationship in Storage between the CMS Interface Module DMSAMS, the CMSAMS DCSS, and the CMSVSAM DCSS

DMSAMS -- Method of Operation

DMSAMS first determines whether the user is in the CMS/DOS environment. If not, a SET DOS ON (VSAM) command is issued to load the CMSDOS segment and to initialize the CMS/DOS environment. In this case, DMSAMS must also issue ASSGN commands for the disk modes in the DOSCB chain created by the OS user's DLBL commands. An ASSGN is also issued for SYSCAT, the VSAM master catalog.

DMSAMS then issues the ASSGN command for the SYSPIT and SYSLOST files, assigning them to the user's A-disk. DLBL commands are then issued associating these units with files on the user's A-disk. Input to the AMSERV processor is in the SYSPIT file. This file has the filetype AMSERV. Output from AMSERV processing is placed in the SYSLST file. This file has the filetype LISTING.

DIAGNOSE code X'64' LOADSYS is then issued to load the CMSAMS DCSS, which contains the VSE/VSAM code. A VSE SVC 65 is issued to find the address of the VSE/VSAM root phase, IDCAMS. When the SVC returns with the address of IDCAMS, a branch is made to IDCAMS, giving control to "live" VSE/VSAM routines.

IDCAMS expects parameters to be passed to it when it receives control. DMSAMS passes dummy parameters in the list labeled AMSPARMS.

After the root phase IDCAMS receives control, the functions in the file specified by the filename on the AMSERV command are executed.

In performing the functions requested in this file, AMS may require execution of VSE/VSAM phases located in the CMSVSAM DCSS. The CMSVSAM DCSS is loaded when ASM opens the VSAM catalog for processing.

On return from VSE/VSAM code, DMSAMS purges the CMSAMS DCSS and issues DLBL commands for the SYSIPT and SYSLST files to clear the DOSCB's for these ddnames.

Control is then passed to DMSVSR, which purges the CMSVSAM DCSS. If the user program was not in the CMS/DOS environment when DMSAMS was entered, the SET DOS OFF command is issued by DMSVSR. Upon return the DMSVSR, DMSAMS performs minor housekeeping tasks and returns control to CMS.

Executing a VSAM Function for a VSE User

When a VSAM function, such as an OPEN or CLOSE macro, is requested from a VSE program, CMS routes control through the CMSDOS DCSS to the CMSVSAM DCSS, thus giving control to VSE/VSAM phases. Figure 27 on page 175 shows the relationships in storage between the user program, the CMSDOS DCSS, and the CMSVSAM DCSS. The description below illustrates the overall logic of that control flow.

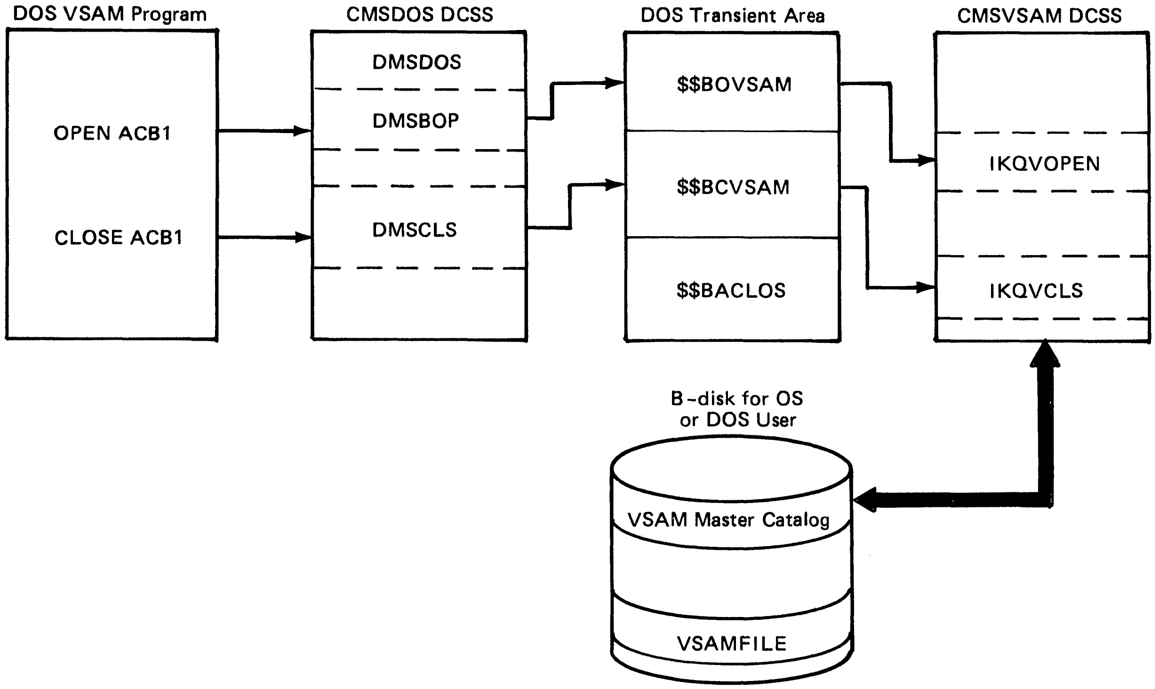


Figure 27. The Relationship in Storage between the User Program, the CMSDOS DCSS, and the CMSVSAM DCSS

CMS/DOS SVC Handling

There are four CMS/DOS routines that handle VSAM requests: DMSDOS, DMSBOP, DMSCLS, and DMSXCP. Within DMSDOS, several SVC functions support VSAM requests. These are described in "Simulating a VSE Environment Under CMS".

DMSDOS VSAM PROCESSING: DMSDOS VSAM processing involves handling of SVC 65 (CDLOAD), which returns the address of a specified phase to the caller. DMSDOS searches both the shared segment table and the nonshared segment table for the CMSDOS and CMSVSAM segments, because both could be in use. Both of these segment tables contain the name of each phase consisting of that segment followed by the fullword address of that phase within the segment.

During SVC 65 processing, DMSDOS checks to see if the IJBLKMD is being requested. IJBLKMD is the VSE lookaside function that VSE/VSAM uses to gain information from the partition anchor tables. If this is the case, DMSDOS returns the address of the IJBLKMD that resides in the CMSBAM DCSS.

If VSAM has not been loaded, a DIAGNOSE code X'64' LOADSYS is issued to load the CMSVSAM DCSS.

DMSBOP VSAM PROCESSING: When DMSBOP is entered to process ACBs, it checks to see if CMSVSAM is loaded. If VSAM has not been loaded, DIAGNOSE code X'64' is issued to load the CMSVSAM DCSS.

DMSBOP then initializes the transient work area and issues a VSE OPEN via SVC 2 to bring the VSAM OPEN $$BOVSAM transient into the VSE transient area.

When VSAM processing completes, control returns to the user program directly.

DMSCLS VSAM PROCESSING: DMSCLS processing is nearly the same as processing for DMSBOP. When DMSCLS is entered, it checks for an ACB to process. If there is one, the $$BCVSAM transient work area is initialized and SVC 2 is issued to FETCH the VSAM CLOSE transient $$BCVSAM into the VSE transient area. When the VSAM CLOSE routines complete processing, control returns to the user program, as in the case of OPEN.

Note: Since VSE does not support the 3380, CMS/DOS cannot access a 3380 when minidisks are formatted as OS/DOS disks.

Executing a VSAM Function for an OS User

OS user requests for VSAM services are handled by VSE/VSAM code that resides in the CMSVSAM DCSS. To access this code, OS VSAM requests are intercepted by the CMS module DMSVIP. DMSVIP is the interface between the OS VSAM requests and the CMS/DOS and VSE/VSAM routines.

Because DMSVIP is in the CMSVSAM segment, it is available only when that segment is loaded. Module DMSVIB, which resides in the CMS nucleus, is a bootstrap routine to load the CMSVSAM segment and to pass control to DMSVIP.

DMSVIP receives control from VSAM request macros in three ways: via SVC (for example, OPEN and CLOSE), via a direct branch using the address of DMSVIP in the ACB, and via a direct branch to the location of DMSVIP whose address is 256 bytes into the CMSCVT. (CMSCVT is a CMS control block that simulates the OS CVT control block.)

This last technique is used by the code generated from the OS VSAM control block manipulation macros (GENCB, SHOWCB, TESTCB, MODCB). That is, the address at 256 into CVT is assumed to be the address of a control block that is at displacement X'12' has the address of the VSAM control block manipulation routine. To ensure that DMSVIP receives control from these requests, the address of DMSVIP is stored at 256 bytes into CMSCVT. However, until the CMSVSAM segment is loaded, the address at CMSCVT+256 is the address of module DMSVIP rather than the address of DMSVIP. The address of DMSVIP replaces that of DMSVIB when CMSVSAM is loaded. Both DMSVIB and DMSVIP have pointers to themselves at 12 bytes into themselves to ensure that this technique works.

Figure 28 on page 177 shows the relationships in storage between the user program, the OS simulation and interface routines, the CMSDOS DCSS, and the CMSVSAM DCSS.

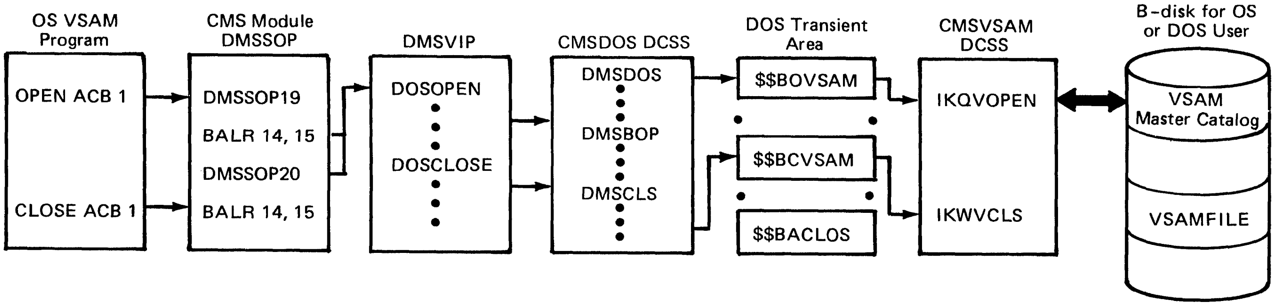


Figure 28. Relationship in Storage between the User Program, OS Simulation and Interface Routines, CMSDOS DCSS, and CMSVSAM DCSS

The following description illustrates the overall logic of that control flow:

DMSVIP Processing

DMSVIP gains control from DMSSOP when an OS SVC 19, 20 or 23 (CLOSE TYPE=T) is issued. It also gains control on return from execution of a VSAM function, as described below. DMSVIP performs five main functions:

- Initializes the CMS/DOS environment for OS VSAM processing
- Simulates an OS VSAM OPEN macro
- Simulates an OS VSAM CLOSE macro
- Simulates an OS VSAM control block manipulation macro (GENCB, MODCB, SHOWCB, or TESTCB)
- Processes OS VSAM I/O macros.

INITIALIZING THE CMS/DOS ENVIRONMENT FOR OS VSAM PROCESSING: DMSVIP gets control when the first VSAM macro is encountered in the user program. Initialization processing begins at this time. The CMSDOS DCSS is loaded by issuing the command SET DOS ON (VSAM). ASSGN commands are also issued at this time according to the user-issued DLBL's indicated in the DOSCB chain. Once this initialization completes, DMSVIP processes the VSAM request.

After the initialization, DMSVIP first checks to determine which VSAM function is being requested, OPEN, CLOSE, or a control block manipulation macro.

Simulate an OS VSAM OPEN

For OPEN processing, the DOSSVC bit in NUCON is set on and control passes to DMSBOP via SVC 2. Once the CMS/DOS routines are in control, execution of the VSAM function is the same as the execution of the VSE/VSAM functions described above.

On return from executing the OPEN routine, the address of another entry point to DMSVIP, at label DMSVIP2, is placed in the ACB for the data set just opened, the DOSSVC bit is turned off, and control is passed to DMSSOP, which returns to the user program. DMSVIP2 is the entry point for code that performs linkage to the VSAM data management phase IKQVSM. This is done after the first OPEN because it is assumed that, once opened, the user performs I/O for the phase; for example, a GET or PUT operation.

When the linkage routine is entered, the DOSSVC bit is set on and control is given to the VSAM data management routine IKQVSM. On return from IKQVSM, DMSVIP turns off the DOSSVC bit and returns control to the user program. (Refer to "Simulate OS VSAM I/O Macros" in this section.)

Simulate an OS VSAM CLOSE

For CLOSE processing, the DOSSVC bit is set on and control is passed to the CMS/DOS routine DMSCLS via SVC 2. As in the case of OPEN, once control passes to the CMS/DOS routine, execution of the VSAM function is the same as the execution of the VSE/VSAM functions described above.

On return from executing the VSAM CLOSE, the DOSSVC bit is turned off and control passes to DMSSOP, which returns to the user program.

SIMULATE OS VSAM CONTROL BLOCK MANIPULATION MACROS: DMSVIP simulates the GENCB, MODCB, SHOWCB, and TESTCB control block manipulation macros.

GENCB Processing: When a GENCB macro is issued with BLK=ACB or BLK=EXLST specified, the GENCB PLIST is passed unmodified to IKQGEN for execution. If GENCB is issued with BLK=RPL and ECB=address specified, the PLIST is rearranged to exclude the ECB specification, because CMS/DOS does not support ECB processing. The GENCB PLIST is then passed to IKQGEN for execution.

MODCB, SHOWCB, and TESTCB Processing: When MODCB, SHOWCB, or TESTCB is issued, the OS ACB, RPL, and EXLST control blocks are reformatted, if necessary, to conform to VSE/VSAM formats.

For MODCB and SHOWCB, the requests are passed to IKQTMS for processing. When MODCB is issued with EXLST= specified, ensure that the exit routines return control to entry point DMSVIP3.

For TESTCB, check for any error routines the user may have specified. If the TESTCB specified RPL= and IO=COMPLETE, a not equal result is passed to the user. All other TESTCB requests are passed to DOS, and the new PSW condition code indicates the results of the test.

If an error return is provided for TESTCB, the address of DMSVIP4 is substituted in the PLIST. This allows DMSVIP to regain control from VSAM so that the DOSSVC bit can be turned off. The error routine is then given control after the address is returned to the PLIST.

SIMULATE OS VSAM I/O MACROS: DMSVIP simulates the OS GET, PUT, POINT, ENDREQ, ERASE, and CHECK I/O macros.

GET, PUT, POINT, ENDREQ, and ERASE Processing: First, the OS request code in register 0 is mapped to a VSE request code. The RPL or chain of RPLs is rearranged to VSE format (unless that has already been done).

If there is an ECB address in the OS RPL, a flag is set in the new VSE RPL and the ECB address is saved at the end of the RPL.

Asynchronous I/O processing is simulated by setting active exit returns inactive in the user EXLST. The exception to this is the JRNAD exit. It need not be set inactive since it is not an error exit. Setting error exits to be inactive prevents VSAM from taking an error exit, thus allowing such an exit to be deferred until a CHECK can be issued for it.

The VSE macro is then issued to IKQVSM via a BALR.

VSE error codes returned in the RPL FDBK field that do not exist in OS are mapped to their OS equivalents. If the user has specified synchronous processing, this return code is passed unchanged in register 15.

For asynchronous processing, return codes are cleared before return and any exist routines set inactive are reactivated in the EXLST. Also, all ECBs are set to WAITING status.

CHECK Processing: For CHECK processing, return codes in the RPL FDBK field are checked to determine the results of the I/O operation. If there is an active exit routine provided for the return code, control is passed to that routine. Also, all WAITING ECBs are posted with an equivalent completion code.

If no active exit routine is provided or if the exit routine returns to VSAM, the return code is placed in register 15 and control is returned to the instruction following the CHECK.

CMS/VSAM ERROR RETURN PROCESSING: Two types of support for error routine processing are provided in DMSVIP. Entry point DMSVIP3 provides support for user exit routines; entry point DMSVIP4 provides support for ERET error returns.

User Exit Routine Processing: DMSVIP provides support for OS VSAM I/O error exits at entry point DMSVIP3. At this entry point the DOSSVC bit is turned off and the user storage key is restored.

The address of the user routine is recovered from VIP's saved exit list (either the primary exit list in the work area or the overflow exit list, OEXLSA).

Control then passes to the appropriate exit routine. If the routine is one that returns to VSAM, the DOSSVC flag is set ON and VSAM processing continues.

DMSVIP can save the addresses of up to 128 exit routines during execution of a user program.

ERET Error Routine Processing: DMSVIP provides support for OS VSAM ERET exit routines used in conjunction with the TESTCB macro. This support is located at entry point DMSVIP4. At DMSVIP4, the DOSSVC bit is turned off and the user storage key is restored. The address of the ERET routine is recovered from the work area and control passes to that routine.

The ERET routine may not return control to VSAM.

Completion Processing for OS and VSE/VSAM Programs

When an OS or VSE/VSAM program completes, control is passed to module DMSVSR, which "cleansup" after VSAM. DMSVSR can be called from three routines after OS processing:

DMSINT if processing completes without system errors or serious user errors

DMSEXT if the user program is used as part of an EXEC file

DMSABN if there are system errors or the user program abnormally terminates

After VSE/VSAM processing completes, DMSVSR is called by DMSDOS.

DMSVSR issues an SVC 2 to execute the DOS transient routine $$BACLOS. $$BACLOS first checks for any OPEN VSAM files. If any are open, SVC 2 is issued to $$BCLOSE (DMSCLS) to close the files.

If there are no open files or if all ACB's have been closed, $$BACLOS issues SVC 2 to $$BEOJ4, an entry point in DMSVSR. At $$BEOJ4, a PURGESYS DIAGNOST 64 is issued to purge the CMSVSAM DCSS. DMSVSR then checks to see if an OS program has completed processing. If this is the case, the SET DOS OFF command is issued and control returns to the caller.

Simulating an OS Environment under CMS

When in a CMS environment, a processor or a user-written program is executing and utilizing OS-type functions, OS is not controlling this action, CMS is in control. Consequently, it is not OS code that is in CMS, but routines to simulate, in terms of CMS, certain OS functions essential to the support of OS language processors and their generated code.

These functions are simulated to yield the same results as seen from the processing program, as specified by OS program logic manuals. However, they are supported only to the extent stated in CMS documentation and to the extent necessary to successfully execute OS language processors. The user should be aware that restrictions to OS functions as viewed from OS exist in CMS.

Certain TSO Service routines are provided to allow the licensed programs to run under CMS. The routines are the command scan and parse service routines and the terminal I/O service routines. In addition, the user must provide some initialization as documented in TSO TMP service routine initialization. The OS functions that CMS simulates are shown in Figure 29.

TSO Service Routine Support

TSO macros that support the use of the terminal monitor program (TMP) service routines are contained in TSOMAC MACLIB. The macro functions are as described in the TSO TMP documentation with the exception of PUTLINE, GETLINE, PUTGET, and TCLEARQ.

Before using the TSO service routines, the calling program performs the following initialization:

1. Stores the address of the command line as the first word in the command processor parameter list (CPPL). The TSOGET macro puts the address of the CPPL in register 1.

2. Initializes CMS storage using the STRINIT macro.

3. Clears the ECT field that contains the address of the I/O work area (ECTIOWA).

4. Issues the STACK macro to define the terminal as the primary source of input.

Macro	SVC No.	Module	Function
XDAP	00	DMSSVT	Reads or writes direct access volumes
EXCP	00	DMSGAM	Executes channel program for graphic access method (GAM)
WAIT	01	DMSSVN	Waits for an I/O completion
POST	02	DMSSVN	Posts the I/O completion
EXIT	03	DMSSLN	Returns from a called phase
RETURN	03	DMSSLN	Returns from a called phase
GETMAIN	04	DMSSMN	Conditionally acquire user storage
FREEMAIN	05	DMSSMN	Releases user-acquired storage
GETPOOL	-	DMSSMN	Simulates as SVC 10
FREEPOOL	-	DMSSMN	Simulates as SVC 10
LINK	06	DMSSLN	Links control to another phase
XCTL	07	DMSSLN	Deletes, then links control to another load phase
LOAD	08	DMSSLN	Reads a phase into storage
DELETE	09	DMSSLN	Deletes a loaded phase

Figure 29 (Part 1 of 3). Simulated OS Supervisor Calls

Macro	SVC No.	Module	Function
FREEMAIN	10	DMSSMN	Manipulates user free storage
GETMAIN	10	DMSSMN	Manipulates user free storage
TIME	11	DMSSVT	Gets the time of day
ABEND	13	DMSSAB	Terminates processing
SPIE	14	DMSSVT	Allow processing program to handle program interrupts
RESTORE	17	DMSSVT	Effective NOP
BLDL	18	DMSSVT	Builds a directory list for a partitioned data set
FIND	18	DMSSVT	Locates a member of a partitioned data set
OPEN	19	DMSSOP	Activates a data file
CLOSE	20	DMSSOP	Deactivates a data file
STOW	21	DMSSVT	Manipulates partitioned directories
OPENJ	22	DMSSOP	Activates a data file
TCLOSE	23	DMSSOP	Temporarily deactivates a data file
DEVTYPE	24	DMSSVT	Obtains device-type physical characteristics
TRKBAL	25	DMSSVT	Effective NOP
FEOV	31	DMSSVT	Sets forced EOV error code
WTO/WTOR	35	DMSSVT	Communicates with the terminal
EXTRACT	40	DMSSVT	Effective NOP
IDENTIFY	41	DMSSVT	Adds entry to loader table
ATTACH	42	DMSSVT	Effective LINK
CHAP	44	DMSSVT	Effective NOP
TTIMER	46	DMSSVT	Accesses or cancels timer
STIMER	47	DMSSVT	Sets timer interval and timer exit routine
DEQ	48	DMSSVT	Effective NOP
SNAP	51	DMSSVT	Dumps specified areas of storage
ENQ	56	DMSSVT	Effective NOP
FREEDBUF	57	DMSSVT	Release a free storage buffer
STAE	60	DMSSVT	Allows processing program to decipher abend conditions
DETACH	62	DMSSVT	Effective NOP
CHKPT	63	DMSSVT	Effective NOP
RDJFCB	64	DMSSVT	Obtains information from FILEDEF command
SYNAD	68	DMSSVT	Handles data set error conditions
SYNADAF	-	DMSSVT	Provides SYNAD analysis function
SYNADRLS	-	DMSSVT	Releases SYNADAF message and save areas
BSP	69	DMSSVT	Backs up a record on a tape or disk
DCB	-	DMSSVT	Constructs a data control block
DCBD	-	DMSSVT	Generates a DSECT for a data control block
SAVE	-	DMSSVT	Saves program registers
RETURN	-	DMSSVT	Returns from a subroutine
GET	-	DMSSQS	Reads system-blocked data (QSAM)
PUT	-	DMSSQS	Writes system-blocked data (QSAM)
READ	-	DMSSBS	Accesses system-record data
WRITE	-	DMSSBS	Writes system-record data
NOTE	-	DMSSCT	Manages data set positioning

Figure 29 (Part 2 of 3). Simulated OS Supervisor Calls

Macro	SVC No.	Module	Function
POINT	-	DMSSCT	Manages data set positioning
CHECK	-	DMSSCT	Verifies READ/WRITE completion
TGET/TPUT	93	DMSSVN	Reads or writes a terminal line
TCLEARQ	94	DMSSVN	Clears terminal input queue
STAX	96	DMSSVT	Creates an attention exit block
PGRLSE	112	DMSSVT	Releases storage contents
CALL	-	-	Transfers control to a control section at a specified entry

Figure 29 (Part 3 of 3). Simulated OS Supervisor Calls

CMS Simulation of OS Control Block Functions

Most of the simulated supervisory OS control blocks are contained in the following two CMS control blocks:

CMSCVT simulates the communication vector table (CVT). Location 16 contains the address of the CVT control section.

CMSCB allocated from system free storage whenever a FILEDEF command or an OPEN (SVC 19) is issued for a data set. The CMS control block consists of the CMS file control block (FCB) for the data file management under CMS, and simulation of the job file control block (JFCB), input/output block (IOB), and data extent block (DEB). The name of the data set is contained in the FCB, and is obtained from the FILEDEF argument list, or from a predetermined file name supplied by the processing problem program.

CMS also utilizes portions of the supplied data control block (DCB) and the data event control block (DECB). The TSO control blocks utilized are the command program parameters list (CPPL), user profile table (UPT), protected step control block (PSCB), and environment control table (ECT).

Operating System Simulation Routines

CMS provides a number of routines to simulate certain operating system functions used by programs such as the Assembler and the FORTRAN and PL/I compilers. The following paragraphs describe how these simulation routines work.

XDAP-SVC 0:
Writes and reads the source code spill file, SYSUT1, during language compilation for PL/I Optimizer and ANS COBOL Compilers.

EXCP-SVC 0:
Executes a channel program. Supported for graphic access method (GAM) only.

WAIT-SVC 1:

Causes the active task to wait until one of more event control blocks (ECBs) have been posted. For each specified ECB that has been posted, one is subtracted from the number of events specified in the WAIT macro. If the number of events is zero by the time the last ECB is checked, control is returned to the user. If the number of events is not zero after the last ECB is checked and the number of events is not greater than the number of ECBs, the active task is put into a wait state until enough ECBs are posted to set the number of events at zero. When the event count reaches zero, the wait bits are turned off in any ECBs that have not been posted and control is returned to the user. If the number of events specified is greater than the number of ECBs, the system abnormally terminates with an error message. All options of WAIT are supported.

POST-SVC 2:

Causes the specified event control block (ECB) to be set to indicate the occurrence of an event. This event satisfies the requirements of a WAIT macro instruction. All options of POST are supported. The bits in the ECB are set as follows:

Bit	**Setting**
0	0
1	1
2-7	Value of specified completion code

EXIT-SVC 3:

This SVC is for CMS internal use only. It is used by the CMS routine DMSSLN to acquire and SVC SAVEAREA on return from an executing program that had been given control by LINK (SVC 6), XTCL (SVC 7) or ATTACH (SVC 42).

GETMAIN-SVC 4:

Control is passed to the GETMAIN entry point in the DMSSMN storage resident routine. The mode is determined: VU, VC, EC. A call is made to GETBLK to obtain the block of storage. Control blocks of two fullwords precede each section of available storage: (1) the address of the next block, (2) the size of this block. The head of the pointer string is located at the words MAINSTRT - initial free block, and MAINLIST - address of first link in chain of free block pointers. All options of GETMAIN are supported except SP, BNDRY=, HIARCHY, LC, and LV.

FREEMAIN-SVC 5:

Releases a block of free storage. If the block is part of segmented storage, a control block of two fullwords is placed at the beginning of the released area. Adjustment is made to include this block in the chain of available areas. All options of FREEMAIN are supported except SP and L.

LINK-SVC 6:

Program transfer is controlled by the nucleus routine, DMSSLN. The LINK macro causes program control to be passed to a designated phase. If the COMPSWT bit within the byte OSSFLAGS is on, loading is done by calling LOADMOD to bring a CMS MODULE file into storage. If this flag is off, dynamic loading is initiated by calling LOAD. If the routine is already in storage, determined by scanning the load request chain, no LOAD or LOADMOD is done. Control is passed directly to the routine. CMS ignores the DCB and HIARCHY options; all other options of LINK are supported.

XCTL-SVC 7:
XCTL first deletes the current phase from storage. Processing then continues as for LINK-SVC 6, as previously described. CMS ignores the DCB and HIARCY options; all other options of XCTL are supported.

LOAD-SVC-8:
Control is passed to DMSSLN8 loaced in DMSSLN when a LOAD macro is issued. If the requested phase is not in storage, a LOAD or LOADMOD is issued to bring it in. Control is then returned to the caller. CMS ignores the DCB and HIARCHY options; all other options of LOAD are supported.

DELETE-SVC 9:
Control is passed to DMSSLN9 located in DMSSLN when a DELETE macro is issued. Upon entry, DELETE checks to see whether the module specified was loaded using LOADMOD or dynamically loaded by LOAD or INCLUDE. If it was loaded by LOADMOD control is returned to the user. If it was dynamically loaded, the responsibility count is decremented by one and if it reaches zero, the storage is released using FREEMAIN, and control is returned to the user. All options of DELETE are supported. Code 4 is returned in register 15 if the phase is not found.

GETMAIN/FREEMAIN-SVC 10:
Control is passed to the SVC 10 entry point in DMSSMN. Storage management is analogous to SVC 4 and 5, respectively. All options of GETMAIN and FREEMAIN are supported. Subpool specifications are ignored.

GETPOOL:
Gets control via an OS LINK macro to IECQBFGI. IECQBFGI allocates an area of free storage using GETMAIN, sets up a buffer control block in the free storage, stores the address of the buffer control block in the DCB, and then returns control to the caller.

TIME-SVC 11:
This routine (TIME) located in DMSSVT receives control when a TIME macro instruction is issued. A call is made (by SIO or DIAGNOSE) to the RPQ software chronological timer device, X'FF'. The real time of day and date are returned to the calling program in a specified form. CMS supports the DEC, BIN, TU, and MIC parameters of the TIME macro instruction. However, the time value that CMS returns is only accurate to the nearest second and is converted to the proper unit.

ABEND-SVC 13:

This routine (DMSSAB) receives control when either an ABEND macro or an unsupported OS/360 SVC is issued. If an SVC 13 was issued with the DUMP option and either a SYSUDUMP or SYSABEND ddname had been defined via a call to DMSFLD (FILEDEF), a SNAP (SVC 51) specifying PDATA=ALL is issued to dump user storage to the defined file. A check is made to see if there are any outstanding STAE requests. If not, or if an unsupported SVC was issued, DMSCWR is called to type a descriptive error message at the terminal. Next, DMSCWT is called to wait until all terminal activity has ceased, and then, control is passed to the ABEND recovery routine. If a STAE macro was issued, a STAE work area is built and control is passed to the STAE exit routine. After the exit routine is complete, a test is made to see if a retry routine was specified. If so, control is passed to the retry routine. Otherwise, control passes to DMSABN unless the task that had the ABEND was a subtask. In that case, the resume PSW in the link block for the subtask is adjusted to point to an EXIT instruction (SVC 3). The EXIT frees the subtask, and the attaching task is redispatched.

SPIE-SVC 14:

This routine (SPIE) receives control when a SPIE macro instruction is issued. When it gets control, SPIE inserts the new program interruption control area (PICA) address into the program interruption element (PIE). The program interruption element resides in the program interruption handler (DMSITP). It then returns the address of the old PICA to the calling program, sets the program mask in the calling program's PSW, and returns to the calling program. All options of SPIE are supported.

RESTORE-SVC 17:

RESTORE is a NOP located in DMSSVT.

BLDL/FIND (Type D)-SVC 18:

SVC to entry points in DMSSOP. If an OS disk is specified, DMSSVT branches and links to DMSROS. See BLDL and FIND under description of BPAM routines in DMSSVT.

STOW-SVC 21:

See STOW under description of BPAM routines in DMSSVT.

OPEN/OPENJ-SVC 19/22:

OPEN simulates the data management function of opening one or more files. It is a nucleus routine and receives control from DMSITS when an executing program issues an OPEN macro instruction. The OPEN macro causes an SVC to DMSSOP. DMSSOP simulates the OPEN macro. The DISP, EXTEND, and RDBACK options are ignored by CMS; all other options of OPEN and OPENJ are supported. You can achieve similar results with the EXTEND option by opening the file with the OUTPUT option and using the DISP MOD parameter on the FILEDEF command.

CLOSE/TCLOSE(CLOSE TYPE=T)-SVC 20/23:

CLOSE and TCLOSE are simulated in the nucleus routine DMSSOP. It receives control whenever a CLOSE or TCLOSE macro instruction is issued. The CLOSE macro causes an SVC to DMSSOP. DMSSOP simulates the CLOSE macro. CMS ignores the DISP option; all other options of CLOSE and TCLOSE are supported.

DEVTYPE-SVC 24:

This routine (DEVTYPE), located in DMSSVT, receives control when a DEVTYPE macro is issued. Upon entry, DEVTYPE moves device characteristic information for the requested data set into a user specified area, and then returns control to the user. All options of DEVTYPE are supported, except RPS, which is ignored.

TRKBAL-SVC 25:

TRKBAL is a NOP located in DMSSVT.

FEOV-SVC 31:

Returns control to CMS with an error code of 4 in register 15.

WTO/WTOR-SVC 35:

This routine (WTO), located in DMSSVT, receives control when either a WTO or a WTOR macro instruction is issued. For a WTO, it constructs a calling sequence to the DMSCWR function program to type the message at the terminal. (The address of the message and its length are provided in the parameter list that results from the expansion of the WTO macro instruction). It then calls the DMSCWT function program to wait until all terminal I/O activity has ceased. Next, it calls the DMSCWR function program to type the message at the terminal and returns to the calling program. All options of WTO and WTOR are supported except those concerned with multiple console support.

For a WTOR macro instruction, this routine proceeds as described for WTO. However, after it has typed the message at the terminal it calls the DMSCRD function program to read the user's reply from the terminal. When the user replies with a message, it moves the message to the buffer specified in the WTOR parameter list, sets the completion bit in the ECB, and returns to the calling program.

EXTRACT-SVC 40:

This routine (EXTRACT), located in DMSSVT receives control when an EXTRACT macro is issued. Upon entry, EXTRACT clears the user provided answer area and returns control to the user with a return code of 4 in register 15.

IDENTIFY-SVC 41:

Located in DMSSVT, this routine creates a new load request block with the requested name and address if both are valid. The new entry is chained from the existing load request chain. The new name may be used in a LINK or ATTACH macro.

ATTACH-SVC 42:

Located in DMSSLN, ATTACH operates like a LINK (SVC 6), with additional capabilities. The user is allowed to specify an exit address to be taken upon return from the attached phase; also, an ECB is posted when the attached phase has completed; and a STAI routine can be specified in case the attached phase abends. The DCB, LPMOD, DPMOD, HIARCHY, GSPV, GSPL, SHSPV, SHSPL, SZERO, PURGE, ASYNCH, and TASKLIB options are ignored; all other options of ATTACH are supported. Because CMS is not a multi-tasking operating system, a phase requested by the ATTACH macro must return to CMS.

CHAP-SVC 44:

CHAP is a NOP located in DMSSVT.

TTIMER-SVC 46:

Checks to ensure that the value in the timer (hex location 50) was set by an STIMER macro. If it was, the value is converted to an unsigned 32 bit binary number specifying 26 microsecond units and is returned in register 0. If the timer was not set by an STIMER macro a zero is returned in register 0, after setting register 0, the CANCEL option is checked. If it is not specified, control is returned to the user. If it is specified, the timer value and exit routine set by the STIMER macro are cancelled and control is returned to the user. All options of TTIMER are supported.

STIMER-SVC 47:

Checks to see if the WAIT option is specified. If so, control is returned to the user. If not, the specified timer interval is converted to 13 microsecond units and stored in the timer (hex location 50). If a timer completion exit routine is specified, it is scheduled to be given control after completion of the specified time interval. If not, no indication of the completion of the time interval is scheduled. After checking and handling any specified exit routine address, control is returned to the user. All options of STIMER are supported. The TASK option is treated as though the REAL option had been specified. The maximum time interval allowed is X'7FFFFF00' timer units (X'00555554' in binary, or 15 hours, 32 minutes, and 4 seconds in decimal). If the time interval is greater than the maximum, it is set to the maximum. If running in the CMSBATCH environment, issuing the STIMER or TTIMER macro will affect the CMSBATCH time limit. Depending on the frequency, number, and duration of STIMER and/or TTIMER issued, the CMSBATCH time limit may never expire.

DEQ-SVC 48:

DEQ is a NOP located in DMSSVT.

SNAP-SVC 51:

Control is passed to SNAP in DMSSVT when a SNAP macro is issued. SNAP fills in a PLIST with a beginning and ending address and calls DMPEXEC. DMPEXEC dumps the specified storage along with the registers and low storage to the printer. Control is then returned to SNAP and SNAP checks to see if any more addresses are specified. It continues calling DMPEXEC until all the specified addresses have been dumped to the printer. Control is then returned to the user. Except for SDATA, PDATA, and DCB, all options of the SNAP macro are processed normally. SDATA and PDATA are ignored. Processing for the DCB option is as follows: The DCB address specified with SNAP is used to verify that the file associated with the DCB is open. If it is not open, control returns to the caller with a return code of 4. If the file is open, the FCB associated with the file is checked for a device type of DUMMY. If the device type is DUMMY, control returns to the caller with a return code of 0 and storage is not dumped.

ENQ-SVC 56:
ENQ is a NOP located in DMSSVT.

FREEDBUF-SVC 57:
This routine (FREEDBUF) located in DMSSVT receives control when a FREEDBUF macro is issued. Upon entry, FREEDBUF sets up the correct DSECT registers and calls the FREEDBUF routine in DMSSBD. This routine returns the dynamically obtained buffer (BDAM) specified in the DECB to the DCB buffer control block chain. Control is then returned to the DMSSVT routine which returns control to the user. All the options of FREEDBUF are supported.

STAE-SVC 60:
This routine (STAE) located in DMSSVT receives control when a STAE macro is issued. Upon entry, STAE creates, overlays or cancels a STAE control block (SCB) as requested. Control is then returned to the user with one of the following return codes in register 15:

Code	**Meaning**
00	An SCB is successfully created, overlaid or cancelled.
08	The user is attempting to cancel or overlay a nonexistent SCB.

Format of SCB

0(0)	0 or pointer to next SCB
4(4)	exit address
8(8)	parameter list address
12(C)	

DETACH-SVC 62:
DETACH is a NOP located in DMSSVT.

CHKPT-SVC 63:
CHKPT is a NOP located in DMSSVT.

RDJFCB-SVC 64:
This routine (RDJFCB) receives control when a RDJFCB macro instruction is issued. When it gets control, RDJFCB obtains the address of the JFCB from the DCBEXLST field in the DCB and sets

the JFCB to zero. It then reads the simulated JFCB located in CMSCB that was produced by issuing a FILEDEF into the closed area. RDJFCB calls the STATE function program to determine if the associated file exists. If it does, RDJFCB returns to the calling program. If the file does not exist, RDJFCB sets a switch in the DCB to indicate this and then returns to the calling program. RDJFCB is located in DMSSVT. All the options of RDJFCB are supported.

- The DCB's specified in the "RDJFCB parameter list" are processed sequentially as they appear in the parameter list.

- On return to the caller, a return code of zero is always placed in register 15 (if an abend occurs, control is not returned to the caller).

- Abend 240 occurs if zero is specified as the address of the area where the JFCB will be placed.

- Abend 240 occurs if a "JFCB exit list entry" (entry type X'07') is not present in the "DCB exit list" for any one of the DCB's specified in the "RDJFCB parameter list."

- If a DCB is encountered in the parameter list with zero specified as the "DCB exit list" ("EXLST") address, RDJFCB immediately returns with return code zero in register 15 -- except if all of the DCB's specified in the "RDJFCB parameter list" are processed unless an abend occurs.

- For a DCB that is not "open," a search is done for the corresponding "FILEDEF" and "DLBL" -- if one is not found, a test is done to determine if a file exists with:

  filename = "FILE"
  filetype = ddname from DCB
  filemode = "A1"

  If such a file does exist, X'40' is placed in the JFCB at displacement X'57' flag "JFCOLD" in field "JFCBIND2"). If such a file does not exist, X'C0' (flag "JFCNEW" will be in field "JFCBIND2").

- For a file that is not "open," but a "DLBL" has been specified, X'08' is placed in the JFCB at displacement X'63' (field "JFCDSORG" byte 2) to indicate that it is a VSAM file.

Note: The switch set by the RDJFCB is tested by the FORTRAN object-time direct-access handler (DIOCS) to determine whether or not a referenced disk file exists. If it does not, DIOCS initializes the direct access file.

SYNAD-SVC 68:

Located in DMSSVT, SYNAD attempts to simulate the functions SYNADAF and SYNADRLS. SYNADAF expansion includes an SVC 68 and a high-order byte in register 15 denoting an access method.

SYNAD prepares an error message line, swap save areas and register 13 pointers. The message buffer is 120 bytes: bytes 1-50, 84-119 blank; bytes 51-120, 120S INPUT/OUTPUT ERROR nnn ON FILE: "dsname"; where nnn is the CMS RDBUF/WRBUF error code. All the options of SYNAD are supported.

SYNADRLS expansion includes SVC 68 and a high order byte of X'FF' in register 15. The save area is returned, and the message buffer is returned to free storage.

BACKSPACE-SVC 69:
Also in DMSSVT. For a tape, a BSR command is issued to the tape. For a direct access data set, the CMS write and read pointers are decremented by one. Control is passed to BACKSPACE in DMSSVT when a BACKSPACE macro is issued. BACKSPACE decrements the read write pointer by one and returns control to the user. No physical tape or disk adjustments are made until the next READ or WRITE macro is issued. All the options of BACKSPACE are supported.

TGET/TPUT-SVC 93:
Located in DMSSVN, this routine receives control when a TGET or TPUT macro is issued. It is provided to support TSO service routines needed by licensed programs. TGET reads a terminal line; TPUT writes a terminal line. The return code is zero if the operation was successful and a four if an error was encountered.

TCLEARQ-SVC 94:
TCLEARQ is located in DMSSVN and causes the terminal input queue to be cleared via a call to DESBUF. At completion a return is made to the user.

STAX-SVC 96:
Located in DMSSVT, STAX gets and chains a CMSTAXE control block for each STAX SVC issued with an exit routine address specified. The chain is anchored by TAXEADDR in DMSNUC. If no exit address is specified the most recently added CMSTAXE is cleared from the chain. If an error occurs during STAX SVC processing, a return code of eight is placed in register 15. The only option of STAX which may be specified is EXIT ADDRESS.

PGRLSE-SVC 112:
Located in DMSSVT, PGRLSE receives control when a PGRLSE macro instruction is issued. The routine checks the validity of the beginning and end addresses of the area to be freed, or forces the right values (AUSRAREA to the beginning, or FREELOWE to the end). Then the routine checks the length of the area to find out if at least 1 page (4K bytes) has to be released and issues a DIAGNOSE code X'10' instruction to CP. The return code will set to zero in register 15 if the PGRLSE operation is successful, or to four if only a portion of the area is released.

CALL:
Transfers control to a control section at a specified entry.

GET/PUT:
See the DMSSQS prolog for description.

READ/WRITE:
OS READ and WRITE macros branch and link to DMSSBS. DMSSBS branches and links to DMSSEB and, if the disk is an OS disk, DMSSEB branches and link to DMSROS. See DMSSBS for description.

NOTE/POINT/FIND (Type C):
OS NOTE, POINT, and FIND (type c) macros branch and link to entry points in DMSSCT. If the disk is an OS disk, DMSSCT branches and links to DMSROS. See DMSSCT for descriptions.

CHECK:
See the DMSSCT prolog for description.

Notes on using the OS simulation routines:

- CMS files are physically blocked in 800-byte blocks and logically blocked according to a logical record length. If the filemode of the file is not 4, the logical record length is equal to the DCBLRECL, and the file must always be referenced with the same DCBLRECL, whether or not the file is blocked. If the filemode of the file is 4, the logical record length is equal to the DCBBLKSI, and the file must always be referenced with the same DCBBLKSI.

- When writing CMS files with a filemode number other than four, the OS simulation routines deblock the output and write it on a disk in unblocked records. The simulation routines delete each 4-byte block descriptor word (BDW) and each 4-byte record descriptor word (RDW) of variable length records. This makes the OS-created files compatible with CMS-created files and CMS utilities. When CMS reads a CMS file with a filemode number other than four, CMS blocks the record input as specified and restores the BDW and RDW control words of variable length records.

  If the CMS filemode number is four, CMS does not unblock or delete BDWs or RDWs on output. CMS assumes on input that the file is blocked as specified and that variable length records contain block descriptor words and record descriptor words.

- To set the READ/WRITE pointers for a file at the end of the file, a FILEDEF command must be issued for the file specifying the MOD option.

- A file is erased and a new one created if the file is opened and all the following conditions exist:

  - The OUTPUT or OUTIN option of OPEN is specified.

  - The TYPE option of OPEN is not J.

- The dataset organization option of the DCB is not direct access or partitioned.

- A FILEDEF command has not been issued for data set specifying the MOD option.

- The results are unpredictable if two DCBs read and write to the same data set at the same time.

Command Flow of Commands Involving OS Access

ACCESS COMMAND FLOW: The module DMSACC gets control first when you invoke the ACCESS command. DMSACC verifies parameter list validity and sets the necessary internal flags for later use. If the disk you access specifies a target mode of another disk currently accessed, DMSACC calls DMSALU to clear all pertinent information in the old active disk table. DMSACC then calls DMSACF to bring in the user file directory of the disk. As soon as DMSACF gets control, DMSACF calls DMSACM to read in the master file directory of the disk. Once DMSACM reads the label of the disk and determines that it is an OS disk, DMSACM calls DMSROS (ROSACC) to complete the access of the OS disk. Upon returning from DMSROS, DMSACM returns immediately to DMSACF, bypassing the master file directory logic for CMS disks. DMSACF then checks to determine if the accessed disk is an OS disk. If it is an OS disk, DMSACF returns immediately to DMSACC, bypassing all the user file directory logic for OS disks. DMSACC checks to determine if the accessed disk is an OS disk; it if is, another check determines if the accessed disk replaces another disk to issue an information message to that effect. Another check determines if you specified any options or fileid and, if you did, a warning message appears on the terminal. Control now returns to the calling routine.

FILEDEF COMMAND FLOW: DMSFLD gets control first when you issue a CMS FILEDEF command. DMSFLD adds, changes, or deletes a FILEDEF control block (CMSCB) and returns control to the calling routine.

LISTDS COMMAND FLOW: The module DMSLDS gets control first when you invoke the LISTDS command. DMSLDS verifies parameter list validity and calls module DMSLAD to get the active disk table associated with the specified mode. DMSLDS reads all format 1 DSCB and if you specified the PDS option and the data set is partitioned, DMSLDS calls DMSROS (ROSFIND) to get the members of the data set. After displaying the DSCB (or DSCB) on your console, DMSLDS returns to the calling routine.

OSRUN COMMAND FLOW: The module DMSOSR gets control first when you invoke the OSRUN command. DMSOSR checks the command syntax. The PARM=parameter, if specified, is set up according to OS convention and a LINK (SVC 6) is issued for the member specified in the OSRUN command. DMSITS (the SVC FLIH) passes control to DMSSVT which in turn goes to DMSSLN for processing of the LINK SVC. DMSSLN passes control to DMSLOS. DMSLOS loads, relocates, and executes the member specified. When the member completes execution and returns control to DMSLOS, DMSLOS returns to DMSSLN for some cleanup; DMSSLN goes through the normal SVC return to DMSOSR. DMSOSR goes through its termination and returns to CMS.

MOVEFILE COMMAND FLOW: The module DMSMVE gets control first when you issue a CMS MOVEFILE command. DMSMVE calls DMSFLD to get an input and output CMSCB and, if the input DMSCB is for a disk file, DMSMVE calls DMSSTT to verify the existence of the input file and gets default DCB parameters in absence of CMSCB DCB parameters. DMSMVE uses OS OPEN, FIND, GET, PUT, and CLOSE macros to move data from the input file to the output file. After moving the specified data, control returns to the calling routine.

LKED COMMAND FLOW: The module DMSLKD gets control first when you invoke a CMS LKED command. DMSLKD generates the necessary FILEDEFs for execution of the OS linkage editor and calls the linkage editor (HEWLFROU). When the link-edit is complete, DMSLKD receives control to do some clean up prior to returning to CMS.

QUERY COMMAND FLOW: The module DMSQRY gets control first when you invoke the QUERY command. DMSQRY verifies parameter list validity and passes control to DMSQRS that calls DMSLAD to get the active disk table associated with the specified mode. DMSQRY displays all the information that you requested on your console. When DMSQRY finishes, control returns to the calling routine.

RELEASE COMMAND FLOW: The module DMSARE gets control first when you invoke the RELEASE command. DMSARE verifies parameter list validity and checks to determine if the disk you want to release is accessed. If the disk you want to release is currently active, DMSARE calls DMSALU to clear all pertinent information associated with the active disk. DMSALU first checks the active disk table for any existing CMS tables kept in free storage. If the disk you want to release is an OS disk, DMSALU does not find any tables associated with a CMS disk. If the disk is an OS disk, DMSALU releases the OS FST blocks (if any) and clears any OS FST pointers in the OS file control blocks. DMSALU then clears the active disk table and returns to DMSARE. DMSARE then clears the device table address for the specified disk and returns to the calling routine.

STATE COMMAND FLOW: The module DMSSTT gets control first when you invoke the STATE command. DMSSTT verifies the parameter list validity and calls module DMSLAD to get the active disk table associated with the specified mode. Upon return from DMSLAD, DMSSTT calls DMSLFS to find the file status table (FST) associated with the file you specified. Once DMSLFS finds the associated FST, it checks to determine if the file resides on an OS disk. If it does, DMSLFS calls DMSROS (ROSSTT) to read the extents of the data set. Upon return from DMSROS, DMSLFS returns to DMSSTT. DMSSTT then copies the FST (or OS FST) to the FST copy in statefst and returns to the calling routine.

OS Access Method Modules--Logic Description

DMSACC MODULE: Once DMSACC determines that the disk you want to access is an OS disk, it bypasses the routines that perform LOGIN UFD and LOGIN ERASE.

If the disk you want to access replaces an OS disk, message DMSACC724I appears at your terminal.

If you specified any options of fileid in the ACCESS command to an OS disk, a warning message, DMSACC230W, appears to notify you that such options or fileid were ignored. DMSACC returns to the calling routine with a warning code of 4.

DMSACF MODULE: DMSACF verifies that the disk you want to access is an OS disk, and, if it is, exits immediately.

DMSACM MODULE: DMSACM saves the disk label and VTOC address in the ADT block if the disk is an OS disk. DMSACM calls the OS access routine (ROSACC) of DMSROS to read the format 4 DSCB of the disk. Upon successful return from DMSROS, control returns to the calling routine. Any other errors are treated as general logon errors.

DMSALU MODULE: If the disk is an OS disk, DMSFRET returns the OS FST blocks (if any) to free storage. DMSALU clears the OS FST pointer in all active OS file control blocks.

DMSARE MODULE: DMSARE ensures that the disk you want to release is an OS disk. DMSARE calls DMSALU to release all OS FST blocks. Upon return from DMSALU, DMSARE clears the common CMS and OS active disk table.

DMSFLD MODULE:

- DSN -- If you specify the parameter DSN as a question mark (?), FILEDEF displays the message DMSFLD220R to request you to type in an OS data set name with the format Q1.Q2.QN. Q1, Q2, and QN are the qualifiers of an OS data set name. If you specify the parameter DSN as Q1.Q2.QN, FILEDEF assumes that Q1, Q2 and QN are the qualifiers of an OS data set name and stores the qualifiers with the format Q1.Q2.QN. in a free storage block and chains the block to the FCB.

- CONCAT -- If you specify the CONCAT option, FILEDEF assumes that the specified FILEDEF is unique unless a filedef is outstanding with a matching ddname, filename, and filetype. This allows you to specify more than one FILEDEF for a particular ddname. The CONCAT option also sets the FCBCATML bit in the FCB to allow the OS simulation routine to know the FCB is for a concatenated MACLIB.

- MEMBER -- If you specify the member option, filedef stores the member name in FCBMEMBR in the FCB to indicate that the OS simulation routine should set the read/write pointer to point to the specified BPAM file member when OPEN occurs.

DMSLDS MODULE: DMSLDS saves the return register, sets itself with the nucleus protection key, clears the dsname key, and initializes its internal flag.

DMSLDS verifies parameter list validity. The data set name must not exceed 44 characters, and the disk mode (the last parameter before the options) must be valid. DMSLDS joins the qualifiers with dots (.) to form valid data set names. If you specify the data set name as a question mark (?), DMSLDS prompts you to enter the dsname in exactly the same form as the dsname which appears on the disk.

DMSLDS calls DMSLAD to find the active disk table block. If you specify filemode as an asterisk (*), DMSLAD searches for all ADT blocks. If you specify the filemode as alphabetic, DMSLAD finds only the ADT block for the specified filemode.

If you specify the dsname (which is optional), DMSLDS sets the channel programs to read by key. If you did not specify a dsname, DMSLDS searches the whole VTOC for format 1 DSCBS and displays all the requested information contained in the DSCB on your console. If you specify the format option, the RECFM, LRECL, BLKSI, DSORG, DATE, LABEL, FMODE, and data set name appear on your console; otherwise, only the FMODE and data set name appear.

If you specify the PDS option, DMSLDS calls the 'find' routine (rosfind) in DMSROS to read the member directory and pass back, one at a time, in the fcbmembr field of CMSCB, the name of each member of the data set. This occurs if the data set is partitioned.

After processing finishes, DMSLDS resets the nucleus key to the same value as the user key, puts the return code in register 15, and returns to the calling routine.

DMSLFS MODULE: DMSLFS verifies that the FST being searched for has an OS disk associated with it. DMSLFS calls the DMSROS state routine (ROSSTT) to verify that the data set exists and CMS supports the data set attributes. Upon return from DMSROS, a return code of 88 indicates that the data set was not found, and DMSLDS starts the search again using the next disk in sequence. Any other errors, such as a return code 80, cause DMSLFS to exit immediately. A return code of 0 from DMSROS indicates that the data set is on the specified disk. From this point on, execution occurs common to both CMS and OS disks.

DMSMVE MODULE: If you specify the PDS option and the input is from a disk, DMSMVE sets the FCBMVPDS bit and issues and OS FIND macro before opening an output DCB to position the input file at the next member. DMSMVE then stores the input member name in the output CMSCB for use as the output filename. After reaching end-of-file on a member, the message DMSMVE225I appears, DMSMVE closes the output DCB, and passes control to find the next member. After moving all the members to separate CMS files, movefile displays message DMSMVE226I, closes the input and output DCBS, and returns control to the calling routine.

DMSROS MODULE:

- ROSACC Routine -- ROSACC gets control from DMSACM after DMSACM determines that the label of the disk belongs to an OS disk. The ROSACC routine reads the format 4 DSCB of the disk to further verify the validity of the OS disk. ROSACC updates the ADT to contain the address of the high extent of the VTOC (if the disk is a DOS disk) or the address of the last active format 1 DSCB (if the disk is an OS disk), and the number of cylinders in the disk. If the disk is a DOS disk, ROSACC sets a flag in the ADT. Information messages appear to notify you that the disk was accessed in read-only mode. If the disk is already accessed as another disk, another information message appears to that effect. Finally, ROSACC zeroes out the ADTFLG1 flag in the ADT, sets the ADRFLG2 flag to reflect that an OS disk was accessed, and returns control to the calling routine.

- ROSSTT Routine -- Verifies the existence of an OS data set and verifies the support of the data set attributes.

  Note: Within the ROSSTT description, any reference to FCB or CMSCB implies a DOSCB if DOS is active.

  ROSSTT gets control from DMSSTT after DMSSTT determines that the STATE operation is to an OS disk. The ROSSTT routine searches for the correct FCB which a previous FILEDEF associated with the data set. If the DOS environment is active, ROSSTT locates the correct DOSCB that defines a data set described by a previous DLBL. If ROSSTT finds an active FST, control passes to ROSSTRET; otherwise, ROSSTT acquires the dsname block, places its address in the FCB, and moves the dsname in the FCB to the acquired block. ROSSTT acquires an FST block, chains it to the FST chain, and fills all general fields (dsname, disk address, and disk mode). ROSSTT now reads the format 1 DSCB for the data set and checks for unsupported options (BDAM, ISAM, VASM, and read protect).

  Errors pass control back to the calling routine with an error code. ROSSTT groups together all the extents of the data set (by reading the format 3 DSCB if necessary) and checks them for validity. ROSSTT bypasses any user labels that may exist and displays a message to that effect. Next, ROSSTT moves the DSCB1 BLKSIZE, LRECL, and RECFM parameters to the OS FST and passes control to ROSSTRET.

- ROSSTRET Routine -- If the disk is not a DOS disk, ROSSTRET passes control back to the caller. If the specified disk is a DOS disk, ROSSTRET fills in the OS FST BLKSIZE, LRECL, and RECFM fields that were not specified in the DSCB1. If the CMSCB fields are zero, ROSSTRET defaults them to BLKSIZE=32760, LRECL=32670, and RECFM=U. Control then returns to the calling routine.

- ROSRPS Routine -- ROSRPS reads the next record of an OS data set. Upon entry to the ROSRPS entry point, ROSRPS calls CHKXTNT and, if the current CCHHR is zero, SETXTNT to ensure the CCHHR and extent boundaries are correctly set. ROSRPS then calls DISKIO and, if necessary, CHKSENSE and GETALT to read the next record. If no errors exist or an unrecoverable error occurred, control returns to the user with either a zero (I/O OK) or an 80 (I/O error) in register 15. If an unrecoverable error occurs, ROSRPS updates the CCWS and buffer pointers as necessary and recalls CHKXTNT and DISKIO to read the next record.

- ROSFIND Routine -- ROSFIND sets the CCHHR to point to a member specified in FCBMEMBR or, if the FCBMVPDS bit is on, sets the CCHHR to point to the next member higher than FCBMEMBR and sets a new member name in FCBMEMBR.

  Upon entry at the ROSFND entry point, ROSFND sets up a CCW to search for a higher member name if the FCBMVPDS bit is on, or an equal member name if the FCBMVPDS bit is off. It then calls SETXTNT, DISKIO and, if needed, CHKSENSE and GETALT to read in the directory block that contains the member name requested. After reading the block, it is searched for the requested member name. If the member name is not found, an error code 4 returns to the calling routine. If an I/O error occurs while trying to read the PDS block, an error code 8 returns to the calling routine. If the member name is found, TTRCNVRT is called to convert the relative track address to a CCHH and pass the address of the member entry to the calling routine.

- ROSNTPTB Routine -- ROSNTPTB gets the current TTR, sets the current CCHHR to the value of the TTR, and backspaces to the previous record.

  Upon entry at the ROSNTPTB entry point, ROSNTPTB checks to determine if a NOTE, POINT, or BSP operation was requested.

  If register 0 is zero, NOTE is assumed. The note routine calls CHRCNVRT to convert the CCHH to a relative track and returns control to the calling routine with the TTR in register 0.

  If register 0 is positive upon entry into DMSROS, POINT is assumed and ROSNTPTB loads a TTR from the address in register 0 and calls TTRCNVRT and SETXTNT to convert the TTR to a CCHHR. Then control returns to the calling routine.

  If register 0 is negative upon entry into DMSROS, BSP (BACKSPACE) is assumed. The backspace code checks to determine if the current position is the beginning of a track. If not, the backspace code decrements the record number by one and control then returns to the calling routine. If the current position is the beginning of a track, the backspace code calls CHRCNVRT to get the current CCHH. The backspace code then calls rdcnt to get the current record number of the last record on the new track, calls setxtnt to set the new extent boundaries, and returns control to the calling routine.

DMSSCT MODULE:

- NOTE Routine -- Upon entry to note, DMSSCT checks to determine if the DCB refers to an OS disk. If it does, DMSSCT calls DMSROS (ROSNTPTB) to get the current TTR. Control then returns to the user.

- POINT Routine -- Upon entry to point, DMSSCT checks to determine if the DCB refers to an OS disk. If it does, DMSSCT calls DMSROS (ROSNTPTB) to reset the current TTR, calls CKCONCAT and returns control to the calling routine.

- CKCONCAT Routine -- Upon entry to CKCONCAT, DMSSCT checks to determine if the FCB MACLIB CONCAT bit is on. If it is on, DCBRELAD+3 sets the correct OS FST pointer in the FCB and returns control to the calling routine. If the FCB MACLIB CONCAT bit is off, control returns to the calling routine.

- FIND (type_C) Routine -- If the DCB refers to an OS disk, DMSSCT calls DMSROS (ROSNTPTB) to update the TTR and control returns to the calling routine.

DMSSEB MODULE:

- EOBROUTN Routine -- If the FCB OS bit is on, control passes to OSREAD. Otherwise, if no special I/O routine is specified in FCBPROC, control passes to EOB2 in DMSSEB.

- OSREAD Routine -- DMSSEB calls DMSROS to perform a read or write and then control passes to EOBRETRN which, in turn, passes control back to DMSSBS. DMSSBS passes control back to the routine calling the read or write macro operation.

DMSSOP MODULE: If the MACLIB CONCAT option is on in the CMSCB, OPEN checks the MACLIB names in the global list and fills in the addresses of OS FSTS for any MACLIBS on OS disks. The CMSCB of the first MACLIB in the global list merges and initializes CMSCBS.

If the CMSCB refers to a data set on an OS disk, DMSSOP checks to ensure that the data set is accessible and the DCB does not specify output, BDAM, or a key length. If any errors occur, error message DMSSOP036E appears and DMSSOP does not open the DCB. DMSSOP fills them in from the OS FST for the data set.

If the CMSCB fcbmembr field contains a member name (filled in by FILEDEF with the member option), DMSSOP issues an OS FIND macro to position the file pointer to the correct member. If an error occurs on the call to the FIND macro, error message DMSSOP036E appears and DMSSOP does not open the DCB.

DMSSVT MODULE:

- BSP (backspace) Routine -- Upon entry, backspace checks for the FCB OS bit. If it is on, the BSP routine calls DMSROS (ROSNTPTB) to backspace the TTR and control returns to the calling routine.

- FIND (type_D) Routine -- Upon entry to find, the find routine checks the FCB OS bit. If it is on, the FIND routine takes the OS FST address from the CMSCB or, if the CONCAT bit is on, from the global MACLIB list. The FIND routine then calls DMSROS (ROSFIND) to find the member name and TTR. DMSROS searches for a matching member name or, if the FCBMVPDS option is specified, a higher member name. If the DMSROS return code is 0 or 8, or if the FCBCATML bit is not on, control returns to the calling routine with the return code from DMSROS. If the return code is 4 and the FCBCATML bit is on, DMSSVT checks to determine if all the global MACLIBS were searched. If they were, control returns to the calling routine with the DMSROS return code. If they were not, DMSSVT issues the FIND on the next MACLIB in the global list.

- BLDL Routine -- BLDL list = FF LL NAME TTR KZC DATA

  If the DCB refers to an OS disk, the BLDL routine fill in the TTR, C-byte and data field from the OS data set.

DMSQRS MODULE:

- SEARCH Routine -- The search routine ensures that any OS disk currently active is included in the search order of all disks currently accessible.

- DISK Routine -- The disk routine displays the status of any or all OS disks using the following form:

  ```
  'MODE(CUU):  (NO. CYLS.), TYPE R/O - OS.'
  ```

DMSSTT MODULE: DMSSTT verifies that the disk being searched is an OS disk. DMSSTT calls DMSLFS to get the FST associated with the data set. Upon return from DMSLFS, DMSSTT checks the return code to ensure that CMS supports the data set attributes. A return code of 81 or 82 indicates that CMS does not support the data set and message DMSSTT229E occurs to that effect. DMSSTT then clears the FST copy with binary zeros, and moves the filename, filetype, filemode, BLKSIZE, LRECL, RECFM, and flag byte to the FST copy. From this point on, common code execution occurs for both CMS and OS disks.

Routines Common to All of DMSROS

- CHRCNVRT Routine -- The CHRNCVRT routine converts a CCHH address to a relative track address.

- CHKSENSE Routine -- CHKSENSE checks sense bits to determine the recoverability of a unit check error if one occurs.

- CHKXTNT Routine -- CHKXTNT checks to determine if the end of split cylinder or the end of extent occurred, and, if so, updates to the next split cylinder or extent.

- DISKIO Routine -- DISKIO starts I/O operation on a CCW string via a DIAGNOSE code X'20'.

- GETALT Routine -- GETALT switches reading from alternate track to prime track and from prime track to alternate track.

- RDCNT Routine -- RDCNT reads count fields on the track to determine the last record number on the track.

- SETXTNT Routine -- SETXTNT sets OSFSTEND to the value of the end of the extent and, if a new extent is specified, sets CCHHR to the value of the start of the extent.

Simulating a VSE Environment Under CMS

CMS/DOS is a functional enhancement to CMS that provides VSE installations with the interactive capabilities of a VM/SP virtual machine. CMS/DOS operates as the background VSE partition; other VSE partitions are unnecessary, since the CMS/DOS virtual machine is a one-user machine.

CMS/DOS provides read access to real VSE data sets, but not write or update access. Real VSE private libraries, system relocatable libraries, source statement libraries, and core-image libraries can be read. This read capability is supported to the extent required to support the CMS/DOS linkage editor, the DOS/PLI, DOS/VS COBOL, and the DOS/VS RPG II compilers, the FETCH routine, and the RSERV, SSERV, and ESERV commands. No read or write capability exists for the VSE procedure library, except for copying procedures from the procedure library (via the PSERV command) or displaying the procedure library (via the DSERV command).

CMS/DOS does not support the standard label area.

Initializing VSE and Processing VSE System Control Commands

Initialization of the CMS/DOS operating environment requires the setting of flags and the creation of certain data areas in storage. Once initialized, these flags and data areas may then be changed by routines invoked by the system control commands.

DMSSET -- Initializing the CMS/DOS Operating Environment

DMSSET initializes the CMS/DOS operating environment as follows:

- Verifies that the mode, if specified, is for a DOS formatted disk

- Stores appropriate data in the SYSRES LUB and PUB

- Locates and loads the CMS/DOS discontiguous shared segment. Saves (in NUCON) the addresses of the two major CMS/DOS data blocks, SYSCOM and BGCOM, and the address of the CMS/DOS discontiguous shared segment (CMSDOS)

- Locates and loads the CMSBAM shared segment if available. This segment contains the following:

  - Simulated VSE OPEN/CLOSE and logic module routines for the VSE sequential access method

- DTFSL support for the DOS PL/I and DOS/VS COBOL compilers
- LBROPEN, LBRFIND, and LBRGET macro simulation as required by the VSE ESERV program
- VSE lookaside function support as required by VSE/VSAM.

- Obtains free storage and initializes the LOCK/UNLOCK resource control table
- Sets the DOSMODE, DOSSVC and CMSBAM bits in DOSFLAGS in NUCON
- Assigns (via ASSGN) the SYSLOG logical unit as the CMS virtual console.

The CMS/DOS operating environment is entered when the CMS SET DOS ON command is issued, invoking the module DMSSET.

Data Areas Prepared for Processing During CMS/DOS Initialization

Several data areas are prepared for processing during initialization. The main CMS data area, NUCON, is modified to contain the addresses of two VSE data areas, SYSCOM and BGCOM. NUCON also contains the address of the Task Control Block (TCB).

The SYSCOM DSECT is the VSE system communications region. It consists mainly of address constants, including the addresses of the boundary box, the PUB ownership table, and the FETCH table. It also includes such information as the number of partitions (always one for CMS/DOS) and the length of the PUB table.

The BGCOM DSECT is the partition communication region. It includes such information as the date, the location of the end of supervisor storage, the end address of the last phase loaded, the end address of the longest phase loaded, bytes used to set the language translator and supervisor options, and the addresses of many other VSE data areas such as the LUB, PUB, NICL, FICL, PIB, and PIB2TAB.

The TCB contains the addresses of the PC and AB exit routines. The TCB also contains the addresses of the related PC and AB exit save area.

The LUB and PUB tables are also made available during initialization. The LUB is the logical unit block table. It acts as an interface between the user's program and the CMS/DOS physical units. It contains an entry for each symbolic device available in the system.

Each of the symbolic names in the LUB is mapped into an element in the PUB, the physical unit block table. The PUB table contains an entry for each channel and device address for all devices physically available to the system and also contains such information as device type code, CMS disk mode, tape mode setting, and 7-track indicator.

Three bits are set in DOSFLAGS in NUCON: DOSMODE, DOSSVC, and CMSBAM. DOSMODE specifies that this virtual machine is running in the CMS/DOS operating environment. DOSSVC indicates whether OS or VSE SVCs are operative in the operating environment. CMSBAM indicates that various VSE functions are supported and available. If DOSSVC is set, VSE SVCs are used. Otherwise, OS SVCs are operative.

Setting or Resetting System Environment Options

Once the CMS/DOS environment is initialized, the flags and control blocks set during initialization can be modified and manipulated to perform the functions specified by commands entered at the console. This section describes the modules that set and reset the system environment options. That is, they set those options that control compiler execution and that control the configuration of logical and physical units in the system.

DMSOPT -- Setting and Resetting Compiler Options

The CMS/DOS OPTION command invokes module DMSOPT, which sets either the default options for the compiler or the options specified on the command line. The nonstandard language translator options switch and the job duration indicator byte are altered. Options are set using two control words located in the partition communication region (BGCOM). Bits in bytes JCSW3 or JCSW4 are set, depending on the options specified.

DMSASN -- Associate System or Programmer Logical Units with Physical Units

module DMSASN is invoked when the ASSGN command is entered. DMSASN first scans the command line to ensure that the logical unit being assigned is valid for the physical unit specified (for example, SYSLOG must be assigned to either the virtual console or the virtual printer). Once the command line is checked, PUB and LUB entries are modified to reflect the specified assignment.

A check is made to ensure that the logical units SYSRDR or SYSIPT are not being assigned to a DOS formatted FB-512 DASD. This is not supported in the CMS/DOS environment because SVC 103 (SYSFIL support) is not available.

For the PUB entry, the device type is determined (via DIAG 24) and the device type code is placed in the PUB. Other modifications are made to the PUB depending on the specified assignment. The LUB entry is then mapped to its corresponding PUB.

DMSDAS -- Dynamically Associated Programmer Logical Units with Physical Units

The function of DMSDAS is to assign a disk device with address X'cuu' to a programmer logical unit (SYS000 - SYS241).

The dynamic assign function supports assigning a DASD unit either permanently or temporarily, changing a DASD unit temporary assign to permanent, or unassigning a DASD. Temporary assigns are cleared either at end-of-job or when the program is cancelled.

DMSDAS first searches the active disk table (ADT) chain to ensure that the X'cuu' supplied is accessed. If the X'cuu' exists, DMSDAS ensures the device is a DASD unit. The programmer LUB table is then searched backwards to find the first available entry. A CMS PLIST is built using the found LUB entry to call DMSASN to actually do the assign.

DMSDAS updates the appropriate LUB entry directly when performing the unassign and change functions.

DMSLLU -- List the Assignments of CMS/DOS Physical Units to Logical Units

The function of DMSLLU is to request a list of the physical units assigned to logical units. It performs this function by referencing information located in the CMS/DOS data blocks, specifically SYSCOM, LUB, and PUB. Another data block, the next in class (NICL) table, is also referenced.

The information on the command line is scanned and the appropriate items are displayed at the user's console. If an option (EXEC or APPEND) is specified, an EXEC file is created ($LISTIO EXEC A1) to contain the output. If EXEC is specified, any existing $LISTIO EXEC A1 file is erased and a new one is created. If APPEND is specified, the new file is appended to the existing file.

DMSDLB -- Associate a DTF Table Filename with a Logical Unit

DMSDLB is invoked when the CMS/DOS DLBL command is entered. DMSDLB associates a DTF (define the file) table filename with a logical unit. This function is performed by creating a control block called a DOSCB, which contains information defining a VSE file used during job execution. DLBL is valid only for sequential or VSAM disk devices.

This information parallels the label information written on a real VSE SYSRES unit under VSE. The DOSCB contains such information as the name, type, and mode of the referenced dataset, its device type code, its logical unit specification, and its dataset type (SAM or VSAM).

A DOSCB is created for each file specified by the user during a terminal session. The DOSCBs are chained to each other and are anchored in NUCON at the field DOSFIRST. The chain remains intact for the entire session, unless an abend occurs or the user specifically clears an entry in the DOSCB chain. A given DOSCB is accessed when an OPEN macro is issued from an executing user program.

The overall logical flow for DMSDLB is as follows:

- Scans the command line to ensure that any options entered are valid (that is, anything to the right of the open parenthesis).

- Processes the first operand (ddname or *). When ddname is specified, loop through the DOSCB chain to find a matching ddname. If none is found, DMSDLB calls DMSFRE to get storage to create a new DOSCB for this file. The old copy of the DOSCB is then saved so that, in case of errors during processing, it can be retrieved intact. The new copy of the DOSCB contains updates, and DOSCB replaces the old copy if there are no errors.

- The mode specification is checked to ensure that it is a valid mode letter; if the file is a CMS file, the mode letter must specify a CMS disk. If DSN has been specified, the mode letter must be for a non-CMS disk.

- Process each option on the command line appropriately.

- If EXTENT or MULT is specified, a separate block of free storage is obtained to contain information about the extent; for example, a block is obtained to contain the VSE data set name.

- Check for errors. If there are errors, any blocks created during processing are purged and an error message is issued. If there are no errors, restore the old block, which has been modified to reflect current processing, and return control to DMSITS.

Process CMS/DOS Open and Close Functions

The CMS/DOS OPEN routines are invoked in response to VSE OPEN macros. They operate on DTF (define the file) tables and ACB (access method control block) tables created when the DTFxx and ACB macros are issued from an executing user program. These tables contain information such as the logical unit specification for the file, the DTF type of the file, the device code for the file, and so forth. The information in the tables varies depending upon the type of DTF specified (that is, the table generated by a unit record DTF macro is slightly different from the table generated by a DTF disk macro).

Five routines are invoked to perform OPEN functions: DMSOPL, DMSOR1, DMSOR2, DMSOR3, and DMBOP. DMSCLS performs the CLOSE function.

OPEN/CLOSE processing in the CMS/DOS environment depends upon the DTF type:

- For DTFCP (disk), DTFDI (disk), and DTFSD DTF types, actual OPEN/CLOSE processing is performed by the simulated VSE SAM routines in the CMSBAM DCSS.

- For all other supported DTF types, OPEN/CLOSE processing is performed totally within the CMS/DOS modules mentioned above.

Opening Files Associated with DTF Tables

Depending on the type of OPEN macro issued from a user program, one of five CMS/DOS OPEN routines could be invoked. OPENR macros give control to DMSOR1, and depending on the DTF type specified, DMSOR2 or DMSOR3 may be invoked. These three routines (DMSOR1, DMSOR2, and DMSOR3) request the relocation of a specified file. DMSOPL is invoked by the VSE compilers when they need access to a source statement library. These routines are mainly interface routines to DMSBOP, which performs the main function of opening the specified file. Each of the routines calls DMSBOP.

DMSBOP is the CMS/DOS routine that simulates the VSE OPEN function for nondisk DTFs. The basic function of DMSBOP for nondisk DTFs is the initialization of DTF tables (that is, setting fields in specified DTFs for use by the VSE LIOCS routines). For disk DTFs, DMSBOP services an interface routine and passes control to the CMSBAM DCSS.

When a VSE problem program is compiling, a list of DTFs and ACBs is built. At execution time, this list is passed to DMSBOP. The logic flow of DMSBOP is as follows:

1. Scans the list of DTF and ACB addresses, handling each item in the list in line. When the OPEN macro expands, register 1 points to the name of the $$B transient to receive control ($$BOPEN) and register 0 points to the list of DTF/ACB addresses to be opened.

2. When an ACB is encountered in the table, control is passed directly to the VSAM OPEN routine, $$BOVSAM. The VSAM routine is responsible for opening the file and returning control to DMSBOP.

3. When a DTF is encountered in the table for nondisk files, DMSBOP itself handles the OPEN:

   a. For reader/punch files (DTFCD), the OPEN bit in the DTF table is turned on.

   b. For printer files (DTFPR), if two IOAREAs are specified, the IOREG is loaded with the address of the appropriate IOAREA. Next, the PUB index byte associated with the logical unit specified in the DTF is checked to ensure that a physical device has been assigned and the PUB device code is then analyzed. The OPEN bit in the DTF table is then turned on.

   c. For console files (DTFCN), no OPEN logic is required.

   d. For tape files (DTFMT), the PUB device type code must specify TAPE. If an IOREG is specified (for output tapes only), the address of the appropriate IOAREA is placed in it. For input files, there is separate processing for tapes with standard label, nonstandard label, and no label. For output tapes, both tape data files and work tape files are treated as no label tapes.

4. For disk files, DMSBOP simulates the function of the VSE transient $$BOSFBL. DMSBOP sets up in the CMSBAM DCSS the input parameters and data areas required by the simulated VSE SAM routines. Control is then passed to the CMSBAM DCSS by placing the address of $IJJGTOP (the SAM OPEN/CLOSE phase) in the problem program save area PSW and exiting via SVC 11.

5. DTFDI and DTFCP are device-independent DTFs. Processing is as above depending upon the type of physical unit to which the DTFs are assigned.

6. If no disk DTFs are encountered, DMSBOP opens all files in the table and returns control to the problem program via SVC 11. If a disk DTF is encountered, DMSBOP exits as described above in step 4 for disk files.

7. If errors are encountered during DMSBOP processing, an error message is issued and return is made via SVC 6.

Closing Files Associated With DTFs

DMSCLS is the CMS/DOS routine that processes CLOSE requests. Its logic is analogous to that of DMSBOP, the OPEN routine described above: when CLOSE expands, register 1 points to $BCLOSE and register 0 points to the list of DTF/ACB addresses. The same table containing DTFs and ACBs used to open files is also used to close those files. Each entry in the table is processed as it occurs, with control passing to a VSAM CLOSE routine ($$BCVSAM) when an ACB is encountered. The OPEN bit is then turned off.

Opening and Closing Files Associated with Disk DTFs

The OPEN and CLOSE functions for disk DTFs are performed by the simulated VSE SAM routines located in the CMSBAM DCSS.

These routines normally issue the LABEL macro to obtain DLBL/EXTENT information from the VSE label area, and issue the OVTOC, PVTOC, and CVTOC macros to obtain VTOC information. These macros require special handling in CMS/DOS. Processing is as follows:

1. DMSLAB (LABEL macro support) -- CMS/DOS does not support the label information area in the same manner as VSE. CMS/DOS keeps similar information in the DOSCB for the file. CMS/DOS intercepts invocations of the LABEL macro and passes control to DMSLAB. DMSLAB obtains the appropriate information from the DOSCB and builds the BLDL/EXTENT record. The DLBL/EXTENT record is then returned to the SAM routines in CMSBAM. Only the GETLBL and GETNXL functions of the LABEL macro are supported. All other functions result in an error return code to the SAM routines in CMSBAM.

2. DMSCVH (OVTOC, PVTOC, and CVTOC macro support) -- In VSE these macros are normally handled by the common VTOC handler routines. These routines are simulated in CMSBAM and are used when

accessing the VTOC on an OS or DOS formatted disk. However, when these macros are issued for a file on a CMS formatted disk, DMSCVH must simulate the appropriate function because CMS formatted disks do not contain a VTOC. VTOC functions simulated by DMSCVH are as follows:

OVTOC - open VTOC
PVTOC - read format 1 label by name
PVTOC - read format 1 label by address
PVTOC - write format 1 label in any slot
PVTOC - write format 1 label by address
PVTOC - check for file overlap
PVTOC - scratch file
CVTOC - close VTOC.

Any other requested VTOC functions is regarded as an error and the program is cancelled via SVC 6.

3. When the SAM routines in CMSBAM complete processing, they exit via an SVC 2 to $$BOSVLT. The functions of this transient are simulated within CMS/DOS by the DMSVLT module. Obtained storage areas are returned and other clean-up functions are performed. DMSVLT exits in one of two different ways:

   - If there are no more DTFs to process, control is returned to the problem program via SVC 11.

   - If there are more DTFs to process, an SVC 2 is issued to the appropriate $$B transient. Then, DMSBOP or DMSCLS is eventually invoked to process the remaining DTFs.

Contents of the CMSBAM DCSS

Several VSE functions are supported within the CMSBAM DCSS as simulated VSE phases. The simulated VSE phases and their functions are as follows:

$IJJGTOP performs OPEN and CLOSE functions for all disk DTFs (DTFSD, DTFDI, and DTFCP).

$IJJHCVH performs VTOC access functions for all disks in DOS format.

$IJBLBSL performs I/O operations to the VSE source statement library for the VSE compilers and the ESERV utility program. The compilers invoke this phase via the DTFSL macro. ESERV invokes this phase indirectly via the LBRFIND and LBRGET macros.

DMSLBR simulates the VSE internal macros LBROPEN, LBRFIND, and LBRGET to the extent required by the VSE ESERV utility program. $IJBLBSL is invoked to perform I/O operations to the VSE source statement library when appropriate.

$IJBLKMD performs the VSE lookaside function as required by VSE/VSAM.

Eight VSE logic modules and two VSE SAM service routines are also simulated as VSE phases. The logic modules handle I/O macros (GET, PUT, POINT, etc.) for SAM files as issued by the user's program. The logic modules and the specific type of SAM file they are associated with are as follows:

$IJGXSDF DTFSD fixed length record data files on DOS formatted FB-512 devices assigned to nonSYSFIL logical units.

$IJGXSDU DTFSD undefined record data files on DOS formatted and CMS formatted disks assigned to nonSYSFIL logical units.

$IJGXSDV DTFSD variable length record data files on DOS formatted FB-512 devices assigned to nonSYSFIL logical units.

$IJGXSDW DTFSD work files on DOS formatted and CMS formatted disks assigned to nonSYSFIL logical units.

$IJGXSVI DTFSD variable length record data files on CMS formatted and DOS formatted FB-512 device, or CMS formatted CKD devices assigned to nonSYSFIL logical units.

$IJGXSFI DTFSD fixed length record data files on CMS formatted and DOS formatted FB-512 device, or CMS formatted CKD devices.

$IJGXCP DTFCP files except for files on DOS formatted FB-512 devices assigned to SYSFIL logical units.

$IJGXDI DTFDI files except for files on DOS formatted FB-512 devices assigned to SYSFIL logical units.

SYSFIL logical units are not supported for use with DOS formatted FB-512 devices in CMS/DOS. SYSFIL logical units refers collectively to logical units SYSRDR, SYSIPT, SYSLST, and SYSPCH.

The SAM service routines issue the actual I/O channel programs for SAM files. The functions they perform are as follows:

$IJGXSSR issues I/O operations for DOS formatted FB-512 devices.

$IJGXSRI issues I/O operations for all CMS formatted disks (FB-512 or CKD) and for DOS formatted CKD devices.

Process CMS/DOS Execution-Related Control Commands

The CMS/DOS FETCH and DOSLKED commands simulate the operation of the VSE fetch routines and the VSE Linkage Editor. The three CMS modules that perform this simulation are:

DMSFET	Provide an interface to interpret the DOS FETCH command line and execute the phase, if START is specified on the command line.
DMSFCH	Bring into storage a specified phase from a system or private core-image library or from a CMS DOSLIB library.
DMSDLK	Link edit the relocatable output of the CMS/DOS language translators to create executable programs.

DMSFET and DMSFCH -- Bring a Phase into Storage for Execution

The VSE FETCH function is simulated by CMS modules DMSFET and DMSFCH. The main control block used during a FETCH operation is FCHSECT, which contains addressing information required for I/O operations.

The FETCH command line invokes module DMSFET. This module first validates the command line and issues a FILEDEF for the DOSLIB file. It then issues a FILEDEF for a DOSLIB file. DMSFET then issues a VSE SVC 4, which invokes the module DMSFCH to perform the actual FETCH operation.

DMSFCH first determines where the phase to be fetched resides. The search order is private core-image library, DOSLIB, system core-image library. If the phase is not found in any of these libraries, DMSFCH assumes that the FETCH is for a phase in a system or private core-image library. To find a DOSLIB library member, OS OPEN and FIND macros are issued (SVC 19 and 18).

When the member is found, OS READ and CHECK macros are issued to read the first record of the file (the member directory). This record contains the number of text blocks and the length of the member.

All addressing information is stored in FCHSECT and the text blocks in that phase are read into storage. If the read is from a CMS disk, issue the OS READ and CHECK macros to read the data. If the read is from a DOS disk, first determine whether this is the first read for the CMS/DOS discontiguous shared segment (DCSS). If this is the case, CCW information is relocated to ensure that the DCSS code is reentrant. For all reads for a DOS disk, a CP READ DIAG instruction is issued. When the entire file is read, it is relocated (if it is relocatable).

If a DOSLIB is open, close it using an OS SVC 20 and return control to DMSFET. DMSFET then checks to see whether START is specified and, if so, an SVC 202 is issued for the CMS START command to execute the loaded file.

When all FETCH processing is complete, control returns to the CMS command handler, DMSITS.

DMSDLK -- Simulate the Functions of the VSE Linkage Editor

CMS simulation of the VSE Linkage Editor function directly parallels the Release 1 implementation of that function. For detailed information on the logic of the function, see *IBM DOS/VSE Linkage Editor Logic*, SY33-8556.

The modules that comprise the VSE Linkage Editor are prefixed by the letters IJB and are separate CSECTs. All of these CSECTs have counterparts contained within the one CMS module, DMSDLK. They are treated as subroutines within that module, but perform the same functions as their independent VSE counterparts and have been named using the same naming conventions as the VSE CSECTs. For example, the IJBESD CSECT in VSE is paralleled by the CMS DMSDLK subroutine DLKESD.

A brief description of the logic follows. The CMS/DOS DOSLKED command invokes the module DMSDLK, which is entered at subroutine DLKINL. DLKINL performs initialization and is later overlaid by the text buffer and the linkage editor tables. DLKINL starts to read from a DOSLNK file and processes ACTION statements, if there are any.

On encountering the first non-ACTION card (or if there is no DOSLNK file), the main flow is entered. Depending on the input on the DOSLNK or the TEXT file, records from either of those files may be read or records from a relocatable library may be read. The type of card image read determines the subroutine to which control is given for further processing.

An ENTRY card indicates the end of the input to the linkage editor. At this point, a map is produced by subroutine DLKMAP. DLKRLD is then entered to finish the editing of object modules by relocating the address constants. If the phases are to be relocatable, relocation information is added to the output on the DOSLIB. Updating of the DOSLIB library is performed by DLKCAT using the OS STOW macro.

A significant deviation from VSE code is the use of OS macros, in some instances, rather than VSE macros. To take advantage of CMS support of partitioned data sets, the OS OPEN, FIND, READ, CHECK, and CLOSE macros are issued rather then their VSE counterparts.

Simulate VSE SVC Functions

All SVC functions supported for CMS/DOS are handled by the following CMS modules:

DMSDOS
DMSETR
DMSGMF
DMSGTM
DMSGVE
DMSLCK
DMSLDF

DMSLIC
DMSMCM
DMSRPG
DMSSTX
DMSSUB
DMSSVL
DMSVIS
DMSXCP

DMSDOS receives control from DMSITS (the CMS SVC handler) when that routine intercepts a DOS SVC code and finds that the DOSSVC flag in DOSFLAGS is set in NUCON.

DMSDOS acquires the specified SVC code from the OLDPSW field of the current SVC save area. Using this code, DMSDOS computes the address of the routine where the SVC is to be handled.

Many CMS/DOS routines (including DMSDOS) are contained in a discontiguous shared segment (DCSS). Most SVC codes are executed within DMSDOS, but some are in separate modules external to DMSDOS. If the SVC code requested is external to DMSDOS, its address is computed using a table called DCSSTABL. If the code requested is executed within DMSDOS, the table SVCTAB is used to compute the address of the code to handle the SVC. Figure 30 on page 213 lists the CMS modules that handle SVC functions supported in CMS/DOS.

Module	Associated SVCs	Function
DMSETR	98	EXTRACT
DMSGMF	107	GETFLD, MODFLD
DMSGTM	34	GETIME
DMSGVE	99	GETVCE
DMSLCK	110	LOCK/UNLOCK
DMSLDF	1 2 4 65	FETCH FETCH LOAD CDLOAD
DMSLIC	50	LIOCS ERRORS
DMSMCM	5	MVCOM
DMSRPG	85	RELPAGE
DMSSTX	16 17 37 95	STXIT PC EXIT PC STXIT AB EXIT AB
DMSSUB	105	SUBSID
DMSSVL	75	SECTVAL
DMSVIS	61 62	GETVIS FREEVIS
DMSXCP	0	EXCP

Figure 30. CMS Modules Handling SVC Functions Supported in CMS/DOS

Figure 31 shows the VSE SVCs and their support in CMS/DOS simulation routines, the name of the macro that invokes a given SVC code, and a brief statement describing how the SVC function is performed.

Function/ Macro	SVC No. Dec	Hex	Support
EXCP	0	0	Used to read from CMS or DOS/OS formatted disk. The CCW's are converted to appropriate CMS I/O requests (for example, RDBUF/WRBUF, CARDRD/CARDPH, etc.). The CCB or IORB is posted according to the CMS return information. DMSDOS will call CMSXCP routine to perform the I/O operation. If a non-zero return code is returned from DMSXCP, a cancel is done. I/O requests to DOS disks are handled using CP DIAGNOSE instructions.
FETCH	1	1	Used to bring a problem program phase into user storage and to start execution of the phase if the phase was found. Operand SYS=YES is not supported. If the user did specify a directory list, a call to DMSFCH is made. Otherwise, DMSDOS will build a directory list using the specified phase name. Once the directory list is prepared, a call to DMSFCH is made. Upon return from DMSFCH, if the phase was found, the entry point address of the phase is saved in the 'SVC' save area oldpsw so that upon return to CMS, DMSITS will then give control to the phase just loaded. If upon return from DMSFCH there were any errors, a cancel is done. If the phase was not found, a message is issued and a cancel is done.
FETCH	2	2	Used to bring a $$B-transient phase into the CMS transient area (or if the phase is in the CMSDOS segment, not to load it), and start execution of the phase if the phase was found. Operand SYS=YES is not supported. A search is made through the loaded segment(s) in an attempt to locate the specified transient. If the phase is found in one of the segments, a call to DMSFCH is not needed. If the phase was not found, a call to DMSFCH is made in a similar way as in SVC 1 above. Once the transient entry point is obtained (from storage or loaded), the address is saved in the SVC save area (as above SVC 1) so that DMSITS gives immediate control to the phase wanted. Errors or not found conditions are handled as above in SVC 1.
FORCE DEQUEUE	3	3	Not supported, see note 2.

Figure 31 (Part 1 of 12). SVC Support Routines and their Operation

Function/ Macro	SVC No. Dec	Hex	Support
LOAD	4	4	Used to bring a problem program phase into user storage, and return the caller the entry point address of the phase just loaded. Operand SYS=YES is not supported. Loading of the requested phase is done exactly as FETCH (SVC 1) calling DMSFCH. Any errors returned from DMSFCH are processed exactly as in fetch. A difference between FETCH (SVC 1) and LOAD (SVC 4) is that upon return from DMSFCH, assuming there are no errors, the user's registers 0 and 1 are updated to contain the address of the directory list (for the user to test if the phase was found), and the entry point address of the phase, respectively. If IJBSIA is being loaded, the address of DMSLAB is returned. If $IJJHCVA (Common VTOC handler) is being loaded, the address of DMSCVH is returned.
MVCOM	5	5	Provides the user with a means of altering positions 12 through 23 of the partition communications region (BGCOM). Before moving the specified information, a test is made to ensure that the range (user's start address, plus length of field to move) will not exceed the allowed range. Once the specified range is found to be within the allowed limits, the user's specified information is moved to the partition communications region.
CANCEL	6	6	Cancels a VSE session either by VSE program request, or by request from any of the CMS routines handling CMS/DOS. Cancel will issue the message `'Job cancelled due to program request'`. A test will be made to see if the value of register 15 upon entry to cancel is below 256. If below, the value in register 15 will be the return code to CMS. If equal or greater, a special return code of 101 will be used to denote that the cancel was issued from a user program (return code of 101 is not used for CMS error messages). Processing then continues using the 'EOJ' code.
WAIT	7	7	Used to wait on a CCB, IORB, ECB, or TECB (note that CMS/DOS does not support ECBs or TECBs). CCBs are always posted by the DMSXCP routine before returning to the caller. The WAIT support under CMS/DOS will effectively be a branch to the CMS/DOS POST routine.

Figure 31 (Part 2 of 12). SVC Support Routines and their Operation

Function/ Macro	SVC No. Dec	 Hex	Support
CONTROL	8	8	Temporarily return control from a $$B-transient to the problem program. If a $$B-transient has to temporarily give control to the problem program, the $$B-transient will issue an SVC 8 passing in register 0 the address of the problem program gaining control. SVC 8 routine will store this address in the SVC work area oldpsw, and return back to CMS SVC handler (DMSITS).
LBRET	9	9	Return to a $$B-transient after an SVC 8 was issued to give control to the problem program. The address saved before (SVC 8 above) is stored in the SVC work area oldpsw, so that when DMSDOS returns to the CMS SVC handler, control is given to the $$B-transient that issued the SVC 8.
SET TIMER	10	A	No operation. Successful return code of 0 is given in register 15. See note 1.
TRANS. RETURN.	11	B	Return from a $$B-transient to the calling problem program. The address saved when the initial SVC 2 (fetch a $$B-transient) was issued, is stored in the CMS's SVC work area oldpsw. Now, when DMSDOS returns to the CMS's SVC handler, control will return to the problem program that issued the SVC 2 calling the $$B-transient.
JOB CTL. 'AND'	12	C	Resets flags to 0 in the linkage control byte in BGCOM (communication region). If register 1 equals 0, bit 5 of JCSW4 (COMREG byte 59) is turned off. If register 1 contains a nonzero value, the function depends on bit 8 of register 1. If bit 8 is zero, this SVC supplies supervisory support to reset flags in the linkage control byte (displacement 57 in BGCOM (communication region)). The user has provided the address of a mask (1 byte) in register 1. An 'AND' operation of the mask with the linkage control byte is performed. If bit 8 is one, this SVC supplies the supervisory support to reset flags in a specified byte of BGCOM (communication region). The user has provided a displacement in byte 2 and a mask in byte 3 of register 1. An 'AND' operation of the mask byte with the specified displacement in the partition communication region is performed.
JC FLAGS	13	D	Not supported. See note 2.

Figure 31 (Part 3 of 12). SVC Support Routines and their Operation

Function/ Macro	SVC No. Dec	 Hex	Support
EOJ	14	E	Normally terminates execution of a problem program. The last SVC save work area is unstacked. Cleanup is done by: 1. Clearing the CMS DOSLIB CMSCB 2. Resetting the JOBNAME in BGCOM 3. Unassigning all temporary device assignments The last return code is loaded into register 15, and control returns to DMSITS (CMSRET).
SYSIO	15	F	Not supported. See note 2.
PC STXIT	16	10	Establish or terminate linkage to a user's program check routine. Locate the appropriate PC option table entry. If the contents of register 0 is zero (terminate linkage), determine if PC routine is active. If the PC routine address in PC option table is negative, terminate linkage by storing zero in routine address field of PC option table. If the routine is not active presently, store zeros in PC routine address field and savearea address field in PC option table. If register 0 is not zero, the address of the PC routine and the savearea address is passed to the STXIT macro. If a STXIT PC routine is active, the complement of the new routine address is placed in the PC option table. If not STXIT PC routine is active, the new PC routine address and savearea address are stored in the PC option table.
PC EXIT	17	11	Used to provide supervisory support for the EXIT macro. SVC 17 provides a return from the user's PC routine to the next sequential instruction in the program that was interrupted due to the program check. Locates the appropriate PC option table entry and restores user's registers and PSW. Stores the address of the PC routine in the PC option table returns to the next sequential instruction in the program that was interrupted.
IT STXIT	18	12	No operation. Successful return code of 0 is given in register 15. See note 1.
IT EXIT	19	13	Not supported. See note 2.
OC STXIT	20	14	No operation. Successful return code of 0 is given in register 15. See note 1.
OC EXIT	21	15	Not supported. See note 2.
SEIZE	22	16	No operation. Successful return code of 0 is given in register 15. See note 1.

Figure 31 (Part 4 of 12). SVC Support Routines and their Operation

Function/ Macro	SVC No. Dec	 Hex	Support
LOAD HEADER	23	17	Not supported. See note 2.
SETIME	24	18	No operation. Successful return code of 0 is given in register 15. See note 1.
HALT I/O	25	19	Not supported. See note 2.
	26	1A	Validate address limits. The upper address must be specified in general register 2 and the lower address must be specified in general register 1. First the lower address must not be negative. An error message DMSDOS005E is issued if it is. Second, the high address cannot be negative. If it is, the same error message is issued. If the low or high address is greater than the end of the partition address in BGCOM, the same error message is issued. Otherwise, control returns to the caller.
TP HALT I/O	27	1B	Not supported. See note 2.
MR EXIT	28	1C	Not supported. See note 2.
WAITM	29	1D	Not supported. See note 2.
QWAIT	30	1E	Not supported. See note 2.
QPOST	31	1F	Not supported. See note 2.
	32	20	Reserved.
COMRG	33	21	Used to provide the caller with the address of the partition communications region. DMSDOS provides the caller with the address of the partition communications region, in the user's register 1.
GETIME	34	22	Provides support for the GETIME macro. SVC 34 updates the date field in the communications region. The GMT operand is not supported.
HOLD	35	23	No operation. Successful return code of 0 is given in register 15. See note 1.
FREE	36	24	No operation. Successful return code of 0 is given in register 15. See note 1.

Figure 31 (Part 5 of 12). SVC Support Routines and their Operation

Function/ Macro	SVC No. Dec	 Hex	Support
AB STXIT	37	25	Establish or terminate linkage to a user's abnormal termination routine. Supported for OPTION=DUMP or NODUMP. Locate the appropriate AB option table entry. If R0 is zero and the AB routine is inactive, then terminate linkage. Otherwise, if the AB routine is active (bit 0 of the AB routine address is on), then cancel the program. If R0 is not zero and the AB routine is active, cancel the program. Otherwise, validate the save area address (must be at least X'20000' and not greater than the partition end), and store the AB routine and save area addresses in the AB option table.
ATTACH	38	26	Not supported. See note 2.
DETACH	39	27	Not supported. See note 2.
POST	40	28	Used to post an ECB, IORB, TECB, or CCB. Byte 2, bit 0 of the specified control block is turned 'on' by DMSDOS.
DEQ	41	29	No operation. Successful return code of 0 is given in register 15. See note 1.
ENQ	42	2A	No operation. Successful return code of 0 is given in register 15. See note 1.
	43	2B	Reserved.
UNIT CHECKS	44	2C	Not supported. See note 2.
EMULATOR INTERF.	45	2D	Not supported. See note 2.
OLTEP	46	2E	Not supported. See note 2.
WAITF	47	2F	Not supported. See note 2.
CRT TRANS	48	30	Not supported. See note 2.
CHANNEL PROG.	49	31	Not supported. See note 2.
LIOCS DIAG.	50	32	Issued by a logical IOCS routine when the LIOCS is called to perform an operation the LIOCS was not generated to perform. The error message 'unsupported function in a LIOCS routine' will be issued, and the session will then be terminated.
RETURN HEADER	51	33	Not supported. See note 2.

Figure 31 (Part 6 of 12). SVC Support Routines and their Operation

Function/ Macro	SVC No. Dec	 Hex	Support
TTIMER	52	34	No operation. Successful return code of 0 is given in register 15. See note 1. Register 0 is also cleared.
VTAM EXIT	53	35	Not supported. See note 2.
FREEREAL	54	36	Not supported. See note 2.
GETREAL	55	37	Not supported. See note 2.
POWER	56	38	Not supported. See note 2.
POWER	57	39	Not supported. See note 2.
SUPVR. INTERF.	58	3A	Not supported. See note 2.
EOJ INTERF.	59	3B	Not supported. See note 2.
GETADR	60	3C	Not supported. See note 2.
GETVIS	61	3D	Used by VSAM to obtain free storage for scratch use or for obtaining an area into which a relocatable VSAM program may be loaded. A free storage subroutine similar to that in the DMSSMN routine is called to obtain the needed space (from the user area). If successful, the address is returned in register 1, and register 15 is cleared. If the request cannot be satisfied, a return code of 12 is passed back in register 15. The PAGE, POOL, and SVA GETVIS option are ignored.
FREEVIS	62	3E	Used to return the free storage obtained via an earlier GETVIS call. The free storage subroutine similar to that in the DMSSMN routine is called to return the area designated by register 1. All complete pages (4K bytes) associated with the returned storage are released by issuing a DIAGNOSE code X'10' to CP.
USE	63	3F	The USE/RELEASE function has been replaced by SVC 110 (LOCK/UNLOCK) for serially controlling system resources. All SVC 63 and 64 requests are mapped into SVC 110 requests respectively. Return codes previously associated with USE/RELEASE under CMS/DOS are maintained.
RELEASE	64	40	Reference SVC 63.

Figure 31 (Part 7 of 12). SVC Support Routines and their Operation

Function/ Macro	SVC No. Dec Hex	Support
CDLOAD	65 41	Used to load a relocatable VSAM phase into storage unless the program has already been loaded. If an anchor table is available, it is searched for the given phase; if found, its load point, entry point, and length are returned in the caller's register 0, 1, and 14 respectively, with register 15 set to 0. If not, DMSFCH is called to find the given phase; if found in a discontinuous shared segment, register 0, 1, and 14 are loaded as above and return made. If the phase was found but is not loaded, storage is obtained (if available) from the GETVIS SVC; DMSFCH is called again to load the program into the storage area just obtained. An anchor table is built in the user area (unless one already exists), the appropriate entries made, and registers 0, 1, and 14 loaded as above, with return to caller. If the program cannot be found, or if storage is unavailable for either loading the program or for building the anchor table, an error code 22 (X'16') is returned to the caller in register 15.
RUNMODE	66 42	Used by a problem program to find out if the program is running in real or virtual mode. The caller's register 0 is zeroed to indicate that the program is running in virtual mode.
PFIX	67 43	No operation. Successful return code of 0 is given in register 15. See note 1.
PFREE	68 44	No operation. Successful return code of 0 is given in register 15. See note 1.
REALAD	69 45	Not supported. See note 2.
VIRTAD	70 46	Not supported. See note 2.
SETPFA	71 47	No operation. Successful return code of 0 is given in register 15. See note 1.
GETCBUF/ FREECBUF	72 48	Not supported. See note 2.
SETAPP	73 49	Not supported. See note 2.
PAGE GIX	74 4A	Not supported. See note 2.

Figure 31 (Part 8 of 12). SVC Support Routines and their Operation

Function/ Macro	SVC No. Dec	 Hex	Support
SECTVAL	75	4B	Used by VSAM I/O routines (ex., IKQIOA) to obtain a sector number for a 3330, 3330-11, 3340, or 3350 device. The appropriate sector value is calculated from the input data supplied by the user's register 0 and 1. If the calculation is successful, the sector number (from 0 to 127) is returned in register 0. If any errors were detected, the noop set-sector value of 255 (X'FF') is returned.
SYSREC	76	4C	Not supported. See note 2.
TRANSCCW	77	4D	Not supported. See note 2.
CHAP	78	4E	Not supported. See note 2.
SYNCH	79	4F	Not supported. See note 2.
SETT	80	50	Not supported. See note 2.
TESTT	81	51	Not supported. See note 2.
LINKAGE	82	52	Not supported. See note 2.
ALLOCATE	83	53	Not supported. See note 2.
SET LIMIT	84	54	Not supported. See note 2.
RELPAGE	85	55	Provides support for the RELPAG macro. At entry register 1 points to a list of 8-byte storage description areas. Each entry contains the beginning address and the length 1 of an area to be released. A nonzero byte following an entry indicates the end of the list. An area is released only if it contains at least a full CP page (4K bytes). Pages are released when the virtual machine calls CP via DIAGNOSE code X'10'. On return, register 15 hold the return code as follows: register 15 = 0 all areas have been released. register 15 = 2 one or more negative area lengths were specified. register 15 = 4 one or more pages t be released were outside the user storage area. register 15 = 16 at least one entry contains a beginning address outside the user storage area.
FCEPGOUT	86	56	No operation. Successful return code of 0 is given in register 15. See note 1.
PAGEIN	87	57	No operation. Successful return code of 0 is given in register 15. See note 1.
TPIN	88	58	Not supported. See note 2.

Figure 31 (Part 9 of 12). SVC Support Routines and their Operation

Function/ Macro	SVC No. Dec	Hex	Support
TPOUT	89	59	Not supported. See note 2.
PUTACCT	90	5A	Not supported. See note 2.
POWER	91	5B	Not supported. See note 2.
XECBTAB	92	5C	Not supported. See note 2.
XPOST	93	5D	Not supported. See note 2.
XWAIT	94	5E	Not supported. See note 2.
AB EXIT	95	5F	Exit from abnormal task termination routine and continue the task. The linkage to either the PC or AB routine is reestablished, and the cancel condition is reset by clearing the ABEND indication in the partition PIB extension. Control is returned to the instruction following the exit AB macro.
TT EXIT	96	60	Not supported. See note 2.
TT STXIT	97	61	Not supported. See note 2.
EXTRACT	98	62	Support for EXTRACT macro of VSE. The caller requests PUB information, CPUID, or storage boundary information. Register 1 on entry points to a PLIST. Output is placed in an area provided by caller.
GETVCE	99	63	Caller requests device information about a specific DASD. Information is returned in an output area pointed to from the PLIST. Register 1 contains a pointer to the PLIST on entry.
	100	64	Reserved.
MODVCE	101	65	No operation. Successful return code of 0 is given in register 15. See note 1.
	102	66	Reserved.
SYSFIL	103	67	Not supported. See note 2.
EXTENT	104	68	No operation. Successful return code of 0 is given in register 15. See note 1.
SUBSID	105	69	SUBSID.. the 'INQUIRY' function is supported for the supervisor subsystem. Information returned is described by the SUPSSID control block. The SUBSID 'NOTIFY' and 'REMOVE' functions are not supported.
LINKAGE	106	6A	Not supported. See note 2.

Figure 31 (Part 10 of 12). SVC Support Routines and their Operation

Function/	**SVC No.**		
Macro	**Dec**	**Hex**	**Support**
TASK INTERF.	107	6B	Provides macro interface support for system information retrieval. The parameters supported are: GETFLD: field=ppsavar returns problem program save area address. =savar returns current save area address. =maintask returns maintask TID in R1. =aclose returns in R1: 1 if in process, 0 if not. =pcexit returns the pcexit routine address and save area in R0 and R1 respectively. If the exit routine is currently active, bit 0 in R0 is set ON. If no exit is defined, it returns a 0 in both R0 and R1. MODFLD: field=vsamopen set bit X'08' in tcbflags byte if R1¬=0 =aclose set bit X'10' in tcbflags byte if R1¬=0 The MODFLD requests for fields CNCLALL and OPENSVA are treated as a NOP with a return code of 0. All other SVC 107 macro calls are unsupported. The error message DMSGMF121S is issued and the request is cancelled. See note 2.
DATA SECURE	108	6C	Not supported. See note 2.
PAGESTAT	109	6D	Not supported. See note 2.
LOCK/ UNLOCK	110	6E	Used by VSAM to control access to resources. Access is maintained in either a 'shared' or 'exclusive' control environment. When DOS is SET ON, counters are maintained as well as the type of control for each resource in a table (LOCKTAB) built in free storage. All entries not unlocked by the program are cleared at both normal and abnormal end-of-job. All requests for resource control are passed to SVC 110 through the DTL (define the lock) macro. SVC 63 requests are mapped into a dummy DTL and processed by SVC 110.

Figure 31 (Part 11 of 12). SVC Support Routines and their Operation

Function/ Macro	SVC No. Dec Hex	Support
		Notes: 1. *No operation:* *In each case, register 15 is cleared to simulate successful operation, and all other registers are returned unchanged, unless otherwise noted.* 2. *Not supported:* *For unsupported SVCs, an error message will be given, and the SVC will be treated as a "cancel."*

Figure 31 (Part 12 of 12). SVC Support Routines and their Operation

Process CMS/DOS Service Commands

DMSSRV	Copies books from a system or private source statement library to a specified output device.
DMSPRV	Copies VSE procedures from a VSE system procedure library to a specified output device.
DMSRRV	Copies modules from a system or private relocatable library to a specified output device.
DMSDSV	Lists the directories of VSE private or system libraries.
DMSDSL	Deletes members (phases) of a DOSLIB library; compresses a DOSLIB library; lists the members (phases) of a DOSLIB library.
ESERV	De-edits, displays or punches, verifies, and updates edit assembler macros from the source statement library.

Terminate Processing the CMS/DOS Environment

DMSBAB	Gives control to an abnormal termination routine once linkage to such a routine has been established via the STXIT AB macro.
DMSITP	Processes program interrupts and SPIE exits.
DMSDMP	Simulates the $$BDUMP and $$BPDUMP routines; issues a CP DUMP command directing the dump to an offline printer.

Chapter 14. Performing Miscellaneous CMS Functions

CMS Batch Facility

The CMS Batch Facility is a function of CMS. It provides a way of entering individual user jobs through an active CMS machine from the virtual card reader rather than from the console. The batch facility reissues the IPL command after each job.

The CMS Batch Facility consists of two modules: DMSBTB, the bootstrap routine (a nonrelocatable CMS module file) and DMSBTP, the processor routine (a relocatable CMS text file that runs free storage).

General Operation of DMSBTB

The bootstrap module, DMSBTB, loads the processor routine DMSBTP and the user exit routines BATEXIT1 and BATEXIT2 (if they exist) into free storage.

DMSBTB first ensures that DMSINS (CMS initialization) has set the BATRUN and BATLOAD flags on in the CMS nucleus constant area indicating that either an explicit batch initial program load command has been issued or that the CMSBATCH command has been issued immediately after initial program load has taken place. (You will receive an error message after issuing the CMSBATCH command if you do not specify the NOSPROF parameter on the IPL command.) If not, error message DMSBTB101E is typed and the batch console returns to a normal CMS interactive environment. STATE (DMSSTT) is then called to confirm the existence of the processor file DMSBTP TEXT. If the file does not exist, error message DMSBTB100E is typed and the batch console returns to the CMS interactive environment.

Using the "state" copy of the file status table (FST) for DMSBTP, DMSBTB computes the size of DMSBTP TEXT file by multiplying the logical record length by the number of logical records (no DS constants). A free storage request is made for the size of DMSBTP, and the address of the routine is then stored at ABATPROC in the NUCON area of the CMS nucleus.

The existence of the user exit routines is determined by STATE. If they exist, their sizes are included in the request for free storage.

The free storage address is translated into graphic hexadecimal format, and the CMS LOAD command is issued to load the DMSBTP TEXT file into the reserved free storage area. The user exit routines, BATEXIT1 TEXT and BATEXIT2 TEXT are also loaded at this time. If these files do not exist, an unresolved external reference error code is returned by the loader, but the error code is ignored by DMSBTB because these routines are optional. If an error (other than unresolved names) occurs, error message DMSBTB101E is typed and the batch console returns to the CMS interactive environment.

The loader tables are searched for the address of the ABEND entry point DMSBTPAB in the loaded batch processor. When the entry is found, its address and that of entry DMSBTPLM are stored in ABATABND and the ABATLIMT respectively, in the NUCON area of the CMS nucleus. If the ABEND entry point is not found in the tables, error message DMSBTB101E is typed and the batch console returns to the CMS interactive environment.

The BATLOAD flag is set off to show that DMSBTP has been loaded, the BATNOEX flag is set on to prevent user job execution until DMSBTP encounters a /JOB card, and finally, control is returned to the command processor DMSINT.

If an error message is issued, DMSERR is called to type the message, and the BATRUN and BATLOAD flags are set off before control is returned to CMS. This allows the normal CMS interaction to resume.

General Operation of DMSBTP

The batch processor module DMSBTP simulates the function of the CMS console read module, DMSCRD. This is accomplished by issuing reads to the virtual card reader, formatting the card-image record to resemble a console record, and returning control to CMS to process the command (or data) request. DMSBTP also performs reads to the console stack if the stack is not empty, checks for and processes the /JOB card, ensuring that it is the first record in the user job, traps all CP commands to maintain system integrity, and performs job initialization, cleanup and job recovery.

Upon receiving control, DMSBTP checks the BATCPEX flag in NUCON. If the flag is set on, control was received from DMSCPF and a branch is made to the CP trap routine to verify that the command is allowable under batch. The function of that routine is described later. If the BATCPEX flag is off, control was received from DMSCRD (console read module) and DMSBTP checks for finished reads in the real batch console stack. If the number of finished reads is not zero, control is returned to DMSCRD to process the real console finished (stacked) reads. If the number of finished reads is zero, a record is read from the batch virtual card reader into the CARD buffer via an SVC call to CARDRD (DMSCIO). The record in the CARD buffer is typed on the console via the WRTERM macro. If the BATMOVE flag is set on (MOVEFILE executing from the console), the records in the file are not typed on the console.

The record in the reader buffer is scanned to compute its length with trailing blanks deleted. It is then moved to the CMS console read buffer, and the computed length is stored in the original DMSCRD parameter list, whose address is passed by DMSCRD when it initially passes control to DMSBTP.

If the first user record is not a /JOB card, error message DMSBTP105E is typed and normal cleanup is performed with the BATTERM flag set on. This flag prevents another initial program load, since it is not needed at this time. Reads to the card reader are then issued until the next /JOB card is found.

If the first record is a /JOB card, DMSBTP branches to its /JOB card processing routine which calls DMSSCNN via a BALR. A check is made for the existence of the userid and account number on the card. If the fields exist, a CP DIAGNOSE code X'4C' is issued to start accounting recording for that userid and account number. If an error is returned from CP denoting an invalid userid or if the userid or account number fields were missing on the /JOB card, error message DMSBTP106E is typed and normal cleanup is performed with the BATTERM flag set on.

The jobname, if provided on the /JOB card, is saved and a message is issued via SVC to inform the source userid that the job has started. The spooling devices are closed and respooled for continuous output, a CP QUERY FILES command is issued for information purposes, and the implied CP function under CMS is disabled and the protection feature set off via SVC calls to SET (DMSSET). The BATPROF EXEC is executed via an SVC to EXEC. The BATNOEX flag, which is set by DMSBTB to suppress user job execution until the /JOB card is detected, is set off. The BATUSEX flag is set on (for DMSCPF) to signal the start of the actual user job, and a branch is taken to read the next card from the reader file (user job).

After reading the /JOB card, DMSBTP continues reading and checks for a /* card, a /SET card, or a CP command. If a card is none of these, DMSBTP passes control back to the command processor DMSINT for processing of the command (or data).

If a /* card is read and it is the first card of the new job, it is assumed to be a precautionary measure and thus ignored by DMSBTP which then reads the next card. If it is not the first card, a check is made for the BATMOVE flag. If the flag is on, the /* card indicates an end-of-file condition for the MOVEFILE operation from the console (reader) and is consequently translated to a null line for the MOVEFILE command.

If the BATMOVE flag is not on, the /* card is an end-of-job indicator and an immediate branch is taken to the end-of-job routine for cleanup and reloading of CMS batch.

When a CP command is encountered, DMSBTP branches to a routine that first checks a table of CP commands allowable in batch. If the command is allowed, a check is made for a reader or other spool device in the command line. If the CP command is allowed but would alter the status of the batch reader or any spooling device or certain disks, or if the command is not allowed at all, error message DMSBTP107E is typed, and the next card is read.

If the CP command is LINK, the device address is stored in a table so that DMSBTP can detach all user disk devices at the end of the job.

A CP DETACH command is examined for a device address corresponding to the system disk, the IPL disk, the batch 195 work disk or any spool device. If the device to be detached is any of these, error message DMSBTP107E is displayed and the next card is read. Otherwise, DMSBTP returns control to DMSINT (or DMSCPF if the BATCPEX flag is set on) for processing of the command.

When a /SET control card is encountered, the card is checked for valid keywords, valid integer values (less than or equal to the installation default values). If an error is detected, error message DMSBTP108E is typed. An abnormal termination message is also sent to the source userid, and the job is terminated with normal cleanup performed. If the control card values are valid, the appropriate fields are updated in the user job limit table DMSBTPLM and the next card is read.

If DMSBTP detects a "not ready" condition at the reader, a message is typed at the console stating that batch is waiting for reader input. DMSBTP then issues the WAITD macro to wait for a reader interrupt. When first detecting the empty reader, DMSBTP calls the CP accounting routines via a CP DIAGNOSE code X'4C' to charge the wait time to the batch userid.

If a hard error is detected at the reader, DMSBTP sends an "intervention required" message to the system console, branches to its abnormal terminal routine, and waits for an interruption for the reader by issuing the WAITD macro.

When a /* card is read (with the BATMOVE flag off) or when the end-of-file condition occurs at the reader, DMSBTP branches to the cleanup routine that sends the source userid a message stating that the job ended normally or abnormally (if cleaning up after an abnormal termination) and turns off the BATUSEX flag (for DMSCPF) to signal the end of the user job. CONWAIT (DMSCWT) is called via SVC to allow any console I/O to finish, the spooling devices are closed (including the console), and all disks that were made available by issuing the CP LINK command are returned by issuing the CP DETACH command.

DMSBTP then relinquishes control by issuing the CP IPL command with the PARM BATCH option which loads a new CMS nucleus, and the next job is started when CMS attempts its first read to the console.

A branch is made to the CMSBTP routine when DMSBTP itself detects an I/O error at the reader. However, the primary purpose of the routine is to receive control not only from DMSABN when there is an abnormal termination during the user job, but also from DMSITE, DMSPIO, and DMSCIO when a user job exceeds one of the batch job limits (BATXLIM flag is on). This routine, entry point DMSBTPAB, calls the CP DUMP routine via SVC, and then it branches to the cleanup routine that reloads

CMS Batch and treats the remainder of the current job as a new job with no /JOB card. This has the effect of flushing the remainder of the job. This technique is used because batch must keep its reader spooled "continuous." Entry point EMSBTPAB is also used by the CMS commands that are disabled in CMS batch. In this case (BATDCMS flag set on), an error message is displayed and control returned to CMS.

When a CP command is called via an SVC in DMSBTP, the CMS CP module (DMSCPF) is actually called to issue the DIAGNOSE instruction to invoke the CP command. DMSBTP calls DMSCPF by issuing a direct SVC 202 or by issuing the LINEDIT macro with the CPCOMM option that generates an SVC 203.

Other CMS Modules Modified in CMS Batch

Several CMS modules check whether CMS batch is running, and, if so, they perform functions associated with batch operation. These modules are shown in the following list:

Module	Function Performed for CMS Batch
DMSINI	Passes batch parameters to DMSINS
DMSINS	Uses batch IPL parameters to reload CMS Batch
DMSLDR	Loads DMSBTP into free storage
DMSCRD	Passes control to DMSBTP to read from the reader rather than from the console
DMSITE	Accounts for virtual time used by batch job -- ABEND if over limit
DMSPIO	Accounts for number of lines printed by batch job -- ABEND if over limit
DMSCIO	Accounts for number of cards punched by batch job -- ABEND if over limit
DMSABN	Passes control to batch ABEND routine in DMSBTP
DMSERR	Passes control to batch ABEND routine instead of entering disabled wait state
DMSMVE	Turns the BATMOVE flag on and off -- allows batch to treat moved blanks as data
DMSSET	Disabled if batch running, except during batch initialization
DMSRDC	Disabled if batch running

DMSCPF	Distinguished between CP command issued by user and by batch
DMSFLD	Disallows reader device specification
DMSDSK	Disk load not allowed in batch

EXEC 2 and System Product Interpreter Processing

Three modules process these functions: DMSEXI, DMSEXE, and DMSREX.

DMSEXI is an interface routine between CMS and either the CMS EXEC interpreter, the EXEC 2 interpreter, or the System Product Interpreter. DMSEXE is the EXEC 2 interpreter. DMSREX is the System Product Interpreter.

A description of each module's method of operation follows.

DMSEXI

MODULE NAME: DMSEXI

CALLED BY: DMSEXC for all EXEC functions

CALLS TO OTHER ROUTINES:

DMSBRD - 'RDBUF' file system function
DMSEXT - CMS EXEC processor
DMSEXE - EXEC 2 processor
DMSFRE - Get and return free storage
DMSREX - System Product Interpreter

EXTERNAL REFERENCES: NUCON, IO

METHOD OF OPERATION:

DMSEXI is an interface routine between CMS and the three EXEC interpreters.

DMSEXI allows coexistence by routing calls to either the System Product Interpreter, the EXEC 2 interpreter, or the CMS EXEC interpreter, according to the following rules:

1. The caller provides an extended-form PLIST, including a file block. DMSEXI directs the call to the EXEC 2 interpreter or System Product Interpreter.

2. The specified EXEC file exists, has a valid format, and contains the character '/*' as the first two non-blank characters in the first 255 characters of the first record or byte 0 of register 1 is X'05'. DMEXI

directs the calls to the System Product Interpreter, after generating a file block and copying or building an extended PLIST.

3. The specified EXEC file exists, has a valid format, and contains the word "&TRACE" within the first 255 bytes of line 1 or byte 0 or register 1 is X'01' or X'0B' and a FILEBLOK exists. DMSEXI directs the calls to the EXEC 2 interpreter, after generating a file block and copying or building an extended PLIST.

4. DMSEXI directs all other cases to the CMS EXEC interpreter, with the original PLIST pointer.

There is one case where DMSEXI must build an untokenized command string to pass to DMSEXE:

- If only a tokenized PLIST is available, DMSEXI builds a command string by concatenating the CMS tokens, separating each by one blank, with no leading or trailing blanks.

DMSEXI releases any storage obtained before it called DMSEXE, then returns to the main caller with the return code from DMSEXE in register 15.

The format of the extended-form PLIST is:

```
PLIST   DS   0F                    (alignment)
        DC   A(parm-string)
        DC   A(byte-following-parm-string)
        DC   A(0) or A(file-block)  (the file to be executed)
```

The following two lines exist only if a function call:

```
        DC   A(arglist)            adlen pairs
        DC   A(funret)             address to store returned data pointer
```

The command-verb and the parm-string form a contiguous area:

```
COMMAND DC   C'command-verb'
        DC   C'parm-string'
```

Trailing blanks are allowed after the command-verb.

The format of the file block is:

```
FILE    DS   0F               (alignment)
        DC   CL8'filename'       (or blank)
        DC   CL8'filetype'       (or blank)
        DC   CL2'filemode'       (eg. A1, or blank)
```

If the filename contains blanks, CMSEXI will use the first word in the argument list (&0) as the filename.

If the filetype contains blanks, DMSEXI will use a filetype of EXEC.

If the filemode is blank, DMSEXI will use the first file with the specified filename and filetype, found according to the file system search order.

The format of the file block extension is:

```
DC   XL2(0000) or XL2(0002)     (number of words in extension)
DC   AL4(PGMFILE)               (address of the in-storage EXEC 2 descriptor)
DC   AL4(PGMEND-PGMFILE)        (number of bytes in descriptor)
```

DMSEXE

MODULE NAME: DMSEXE

CALLED BY: DMSEXI to interpret EXEC 2 statements

CALLS TO OTHER ROUTINES:

DMSERR - Write all error messages
DMSSCN - Tokenize strings
DMSPNT - ‘POINT’ file system function
DMSSTT - ‘STATE’ file system function
DMSBRD - ‘RDBUF’ file system function
DMSFNS - ‘FINIS’ file system function
DMSFRE - Get and return free storage
WAITRD - Read from the terminal
TYPLIN - Type on the terminal
ATTN - Stack lines in console stack

EXTERNAL REFERENCES: NUCON, FST, FVS, ADT

METHOD OF OPERATION:

DMSEXE reads lines from disk files, or accepts lines previously prepared by the caller and stored in main memory.

If the lines are EXEC 2 statements, DMSEXE interprets the statements. If the lines contain commands, DMSEXE passes the commands to CMS command mode or a subcommand environment.

Execution continues until a statement or command explicitly terminates it, or DMSEXE finds a statement error.

DMSEXE LOGIC DESCRIPTION:

Pseudo Code

Pseudo code is used to describe the logic of portions of DMSEXE. This pseudo code has the following general statements:

1.

```
DO
    statement
    statement
     ...
    statement
    END
```

"Statement" is either:

a. A description of an action to be done, or

b. Another pseudo code statement:

```
DO...END,
IF...THEN...ELSE,
GOTO, or
CALL
```

2. If condition THEN statement ELSE statement.

   "Condition" is a hyphenated sequence of words describing the conditions for which the statement after "THEN" is executed.

   ```
   Example: IF initial-flag-is-set
                THEN perform initialization
                ELSE indicate error condition
   ```

3. GOTO label

   Transfer control to the label specified. A label is followed by a colon and precedes a statement, or is on a line by itself.

   ```
   Example: GOTO George
                ...
                George: ...
   ```

4. CALL name

   CALLs the named subroutine.

DMSEXE General Logic Flow

After initialization, DMSEXE loops continually, reading lines that may contain EXEC 2 statements or commands. The logic follows:

```
Initialization
DO forever
  Loop initialization
  IF executing &loop
    THEN DO
        Test condition
        IF condition
          THEN set for top of loop
          ELSE set for exit from &loop
        END
```

```
  CALL READSUB   (read next line)
  IF eof
    THEN IF executing-&loop
           THEN error condition
           ELSE exit
  CALL EXECUTE   (execute line)
END
```

READSUB/READLAB

READSUB is the DMSEXE subroutine that reads the next line. READLAB, a secondary entry point to READSUB, reads the next line when scanning forward for labels.

READSUB reads a line from:

1. The console - if the console count is non-zero,
2. The cache - if there is one, and the needed line is there,
3. 'BUF' - if the needed line is there, or
4. The file - if none of the above conditions are true.

If the line is read from the file, and there is a cache, then the line is read into the cache.

READLAB reads a line from:

1. The cache - if there is one, and the line is there,
2. 'BUF' - if the line is there, or
3. The file - if none of the above conditions are true.

If the line is read from the file, and there is a cache, then the line is read into the cache.

In all cases:

1. A blank and a zero byte are placed at the end of the line.
2. The file read may be either an in-storage file, or a file accessed by calls to file system routines.

Line Execution

DMSEXE executes lines according to the following logic:

```
EXECUTE: IF comment THEN exit

    IF tracing THEN trace the line

    IF blank-line THEN exit

    IF assignment
        THEN DO
            CALL ASSIGN    (perform assignment)
```

```
                    Exit
                  END
            IF command
                THEN DO
                    Pass command to CMS command mode or
                        subcommand environment
                    Exit
                  END
            (Line must be a control-statement:)
            Look up control-statement word
            IF found
                THEN DO
                    GOTO control-statement routine:
                        ex. ARGS
                            BEGPRINT
                            BEGSTACK
                            BUFFER
                             ...
                     Exit
                   END
                ELSE error (invalid statement)
      END EXECUTE
```

Assignment Processing

DMSEXE processes assignment statements according to the following logic:

```
    ASSIGN:  CALL SUBS  (Substitute value of EXEC variable into
                   characters 2 through N of target)
          Point to first word after equal sign
          Call GETNEXT
          IF none THEN set null value and exit
          Call GETNEXT
          IF none THEN set value obtained above and exit

     Top-of-loop:
           IF last-word-is-not-an-operation
               THEN error
           Call GETNEXT
           IF none THEN error
           Call GETNEXT
           IF none
               THEN DO
                   Do calculation
                   Set value
                   Exit
                 END
           IF function-reference
               THEN DO
                   IF not 'of' THEN error
                   IF system-function
                     THEN Call appropriate routine
                         to evaluate function
                     ELSE invoke user function
                  Exit
                END
           Do calculation
     GOTO Top-of-loop
  END ASSIGN

    GETNEXT:  Get next word
        IF found
            THEN DO
                Call SUBS
                IF null THEN GOTO GETNEXT
             END
  END GETNEXT

    SUBS:  Set pointer to end of word plus one

    SUBSLP:
        Decrement pointer
        IF at-front-of-word THEN exit
        IF not '&' THEN GOTO SUBSLP
        Calculate hash using last character of name and length
        Scan appropriate variable lookaside chain
        IF found
            THEN DO
                IF not-at-front-of-chain THEN put at front
                IF predefined-variable
                    THEN DO
                        CALL predefined variable routine
                            and substitute value
                        IF at-front-of-word THEN exit
                        GOTO SUBSLP
                      END
                Substitute value
                IF at-end-of-word THEN exit
                GOTO SUBSLP
              END
            ELSE DO
                IF predefined-name
                    THEN DO
                        Build variable blocks
                        Point block to processing routine
                        CALL routine and substitute value
                      END
                    ELSE DO
                        Build variable block for null
                            value
                        Substitute null
                      END
                IF at-front-of-word THEN exit
                GOTO SUBSLP
         END
END SUBS
```

DMSREX

DMSEXI sends EXEC files (files written in the Restructured Extended Executor (REXX) language) to DMSREX (System Product Interpreter) if the first two non-blank characters in the first 255 characters of the first record are '/*'. All System Product Interpreter processing is done by DMSREX. DMSREX has the following CSECTS:

DMSRCN Performs character conversion, console I/O, general services, and all arithmetic

DMSREV Evaluates expressions

DMSRFN Performs all built-in functions

DMSRIN Parses input data, controls most execution decisions, and passes clauses to DMSRXE for execution

DMSRTC Formats and displays trace information

DMSRVA Accesses and maintains variables

DMSRXE Executes individual clauses

DMSREX Reads the EXEC file and calls DMSRIN

DMSRSF Performs additional functions similar to the built-in functions

Part 3: CMS Directory

This part contains the following information:

- Module Entry Point Directory

Chapter 15. Module Entry Point Directory

Module Name	Entry Points	Function
DMSABN	DMSABN	Intercepts an abnormal termination (ABEND) and provides recovery from the ABEND. Entered by a DMKABN TYPCALL=BALR macro call.
	DMSABNKX	Entered by a KXCHK macro to halt execution after HX has been entered after signaling attention.
	DMSABNGO	Entered by any routine that sets up ABNPSW and ABNREGS in the work area beforehand.
	DMSABNXV	Entered as the result of a DMSABN TYPCALL=SVC macro call.
	DMSABNRT	Returns entry point from DEBUG.
DMSABX		Receives control when the ABNEXIT macro is executed.
	DMSABX	Handles the SET or CLEAR attribute of the ABNEXIT macro.
	DMSABXR	Handles the RESET attribute of the ABNEXIT macro.
DMSACC	DMSACC	ACCESS command, parameter checking: checks parameters, issues error messages, sets up parameter list to prepare for DMSACP to perform the actual access.
DMSACF	READFST	Reads all file status table blocks into storage for a read/write disk. Reads in file management tables for a R/O disk. For an O/S disk, control returns to the caller after a successful return from DMSACM.
DMSACG	DMSACG	For an EDF disk, not a S- or Y-disk, reads all the FST entries by hyperblocks into storage using RDBUF. The FSTs are sorted, if necessary, using a Quicksort algorithm. A hyperblock structure is rebuilt in storage to reflect the files to be ACCESSed for this disk.
DMSACM	READMFD	Reads the ADT, QMSK, QQMSK, and first chains link into virtual storage from the master file directory on disk.
DMSACP	DMSACP	ACCESS command: brings the user file directory for a given disk (e.g. 191, 192) into storage, setting up the necessary information in the active disk table for the given disk mode letter.
DMSACS	DMSACS	For an EDF disk accessed R/O for all filenames and filetypes, DMSACS loads a DCSS containing the file status table blocks, if they are available.
DMSADD	DMSADD	Adds a server to a list of servers to be notified when communications end.

Module Name	Entry Points	Function
DMSALA	DMSALA	Processes the ALARM command.
DMSALC	DMSALC DMSALCU DMSALCEC DMSALCCB DMSALCBC	Allocates user memory. Frees allocated user memory. Issues an SVC 202 (02) to EXECCOMM subcom. Converts EBCDIC character string to binary number. Converts binary number to EBCDIC character string.
DMSALP	DMSALP	Marks the start of the CMS nucleus code.
DMSALU	RELUFD SORTFST	For a specified disk, releases all tables kept in free storage and clears appropriate information in the active disk table (ADT). Also purges the DCSS if the file status table blocks were in a DCSS. Sorts FST entries for a CMS disk under some conditions.
DMSAMS	DMSAMS	Provides an interface to VSE/VSAM Access Method Utility programs (IDCAMS). Provided for support of CMS/VSAM.
DMSARD	DMSARD	Provides storage for the ASM3705 assembler auxiliary directory. DMSARD contains no executable code. It must be loaded with DMSARX and the GENDIRT command must then be issued to fill in the auxiliary directory entries. GENMOD must then be issued to create the ASSEMBLE module.
DMSARE	DMSARE	Releases storage used for tables pertaining to a given disk when that disk is no longer needed.
DMSARN	DMSARN ASMHAND	This is the ASM3705 command processor. It provides the interface between user and the 370x Assembler. This is the SYSUT2 processing routine called from DMSSOB and used during the assembly whenever any I/O activity pertains to the SYSUT2 file.
DMSARX	DMSARX	Provide an interface for the ASM3705 command to the 3705 assembler program.
DMSASD	DMSASD	Provides storage for the assembler auxiliary directory. DMSASD contains no executable code. It must be loaded with DMSASM and the GENDIRT command must then be issued to fill in the auxiliary directory entries. The GENMOD command must then be issued to create the assemble module.
DMSASM	DMSASM ASMPROC	Processes the ASSEMBLE command. Provides the interface between the user and the system assembler. This is the SYSUT1 processing routine (called from DMSSOB).
DMSASN	DMSASN	Associates logical units with a physical hardware device. (Interface for the ASSGN command used by CMS/DOS and CMS/VSAM.)
DMSAST	DMSAST	Gets a new CPRB, copies the one created by the requester, and issues the SVC to the server with the copied CPRB.

Module Name	Entry Points	Function
DMSAUD	DMSAUD DMSAUDUP	Reserves space on disk for writing a copy of disk and file management tables on disk and then updates the master file directory. When called by non-resident routines, attempts to insure that the R/W disk hash table has been updated with the new file (if any). Closes all CMS files, thereby updating the master file directory for any disks that had an output file open.
DMSBAB	DMSBAB	Give control to an abnormal termination routine once linkage to such a routine has been established by STXIT AB macro.
DMSBCT	DMSBCT	Handles broadcast to all the servers on the list that are notified when communications end.
DMSBLG	DMSBLG	Builds the COPROC panel.
DMSBOP	DMSBOP	Opens CMS/DOS files associated with the following DTF (define the file) tables: DTFCN, DTFCD, DTFPR, DTFMT, DTFDI, DTFCP, DTFSD. For nondisk files, the OPEN function is performed in its entirety by DMSBOP. For disk files, the SAM OPEN/CLOSE routines in CMSBAM are invoked. Once the files are opened and initialized, I/O operations can be performed using the file.
DMSBOR	DMSBOR	Processes the border commands.
DMSBRD	DMSBRD (RDBUF)	Reads one or more successive items from a specified file. DMSBRD, itself, reads items from 800-byte formatted disks, or calls DMSERD at the DMSERDBF entry point to read items from 512-, 1K-, 2K-, or 4K-byte formatted disks.
DMSBSC	BASIC	Processes the BASIC command. The BASIC command invokes the CALL-OS BASIC language processor to compile and execute the specified file of BASIC source code.
DMSBTB	DMSBTB	This is the CMS batch bootstrap routine. It loads the batch processor routine (DMSBTP) and user exit routine (if they exit) into free storage.
DMSBTP	DMSBTP DMSBTPAB DMSBTPLM	Main entry; reads from the virtual card reader each time CMS tries to execute a console read. Entry point for abnormal conditions during user job: ● Job execution ABEND (from DMSABN) ● Job limit exceeded (from DMSITE, DMSCIO, DMSPIO) ● Disabled CMS command (from the command) Non-executable user job limit table referenced by DMSITE, DMSPIO, and DMSCIO.
DMSBWR	DMSBWR	Writes one or more successive items into a specified disk file. DMSBWR, itself, writes to 800-byte formatted disks, or calls DMSERD at the DMSEWRBF entry point to write items to 512-, 1K-, 2K-, 4K-byte formatted disks.

Module Name	Entry Points	Function
DMSCAT	DMSCAT	Stacks a line of console input that DMSCRD reads later when it is called.
	DMSCATMK	MAKEBUF command.
	DMSCATNB	SENTRIES command.
DMSCCK	CATCHECK	Provides an interface to the VSE/VSAM Catalog Check Service Aid. Provided for support of CMS/VSAM.
DMSCCP	DMSCCP	Returns the pointer to the top CPRB in the CPRB stack.
DMSCCS	DMSCCSVC	Console Communication Service module handles all fullscreen console I/O functions requested via the CONSOLE macro.
	DMSCCSVR	Handles reconnect processing. Entered from DMSITI (I/O interrupt handler) when a reconnect of a previously disconnected device occurred.
DMSCDI	DMSCDI	Resolves and issues CMS commands from the command lines.
DMSCIO	DMSCIOR	Reads one card record.
	DMSCIOP	Punches one card record.
	DMSCIOSI	Punch caller's buffer.
DMSCIT	DMSCIT	Processes the interruptions for all CMS terminal I/O operations and starts the next I/O operation upon completion of the current I/O operation.
	DMSCITA	Processes terminal interruptions.
	DMSCITB	Starts next terminal I/O operation.
	DMSCITDB	Frees I/O buffers from stacks.
	DMSCITDK	DROPBUF command.
DMSCLN	DMSCLN	Zeros out the user word in the SCBLOCK of the SRPI subcommand. DMSCLN is called on CMS abend after all control blocks have been released.
DMSCLR	DMSCLR	Processes the CLEAR command.
DMSCLS	DMSCLS	Closes CMS/DOS files associated with the following DTF (define the file) tables: DMTCN, DTFCD, DTFPR, DTFMT, DTFDI, DTFCP, and DTFSD. For nondisk files, the CLOSE function is performed in its entirety by CMSCLS. For disk files, the VSE OPEN/CLOSE routines in CMSBAM are invoked.
DMSCMP	COMPARE	Compares the records contained in two disk files.
DMSCPF	DMSCPF	Passes a command line to CP for execution.
DMSCPY	DMSCPY	Processes the COPYFILE command to copy disk files.
DMSCRD	DMSCRD	Reads a line of console input.
	DMSCRDCN	Performs a linemode read from the console. Called by LINERD (DMSWLR).
	DMSCRDSK	Reads a line from the program stack. Called by LINERD (DMSWLR).
DMSCRT	DMSCRT	Obtains storage for a new session header, initializes it, and chains it to the list structure anchor.
DMSCSF	DMSCSF	Contains the subcom environment (CMS) for the command search function.

Module Name	Entry Points	Function
DMSCST	DMSCST	Puts the pointer to the current CPRB on the CPRB stack.
DMSCUR	DMSCUR	Processes the CURSOR command.
DMSCVH	DMSCVH	Simulates VTOC functions for CMS formatted disks in the CMS/DOS environment.
DMSCWR	DMSCWR	Writes an output line to the console.
DMSCWT	DMSCWT	Causes the calling program to wait until all terminal I/O operations have been completed.
DMSDAS	DMSDAS	Simulates the VSE ASSIGN macro.
DMSDBD	DMSDBD	Enables a user to dump his virtual storage from within an executing program.
DMSDBG	DMSDBG DMSDBGP DMSDBG	Enables the user to debug his program from the terminal. Entry point for program interruptions. Entry point for all other interruptions.
DMSDDL	DMSDDL	Send files in NETDATA form to a user on a local or a remote node. Receive and query NETDATA files in user's reader.
DMSDEF	DMSDEF	Processes the DEFINE command.
DMSDEL	DMSDEL	Processes the DELETE command.
DMSDEV	DMSDEV	Processes the CMSDEV macro. Obtains VM/SP device characteristic information and places it in a user-provided buffer.
DMSDFT	DMSDFT	Provides IBM Cooperative Processing communications with the work station. Initializes itself, establishes communications with the work station, receives work station requests and forwards them to a named server, receives server replies and forwards them to the work station.
DMSDID	DMSDID	Returns the virtual address, blocksize, and offset of a RESERVEd mini-disk to the user.
DMSDIO	DMSDIOR DMSDIOW	Reads one or more 800-byte records (blocks) from disk, or reads one 200-byte record (sub-block) from disk. Writes one or more 800-byte records (blocks) on disk, or writes one 200-byte record (sub-block) on disk.
DMSDIP	DMSDIPOS DMSDIPDI DMSDIPBI DMSDIPTI DMSDIPFB	VM to OS Count Key Data device code conversion table. Index to device constants by device type. Index to device constants by blocksize. Constants table address table. VM to OS Fixed Block Architecture device code conversion table.
DMSDLB	DMSDLB	Interface for the CMS/DOS DLBL command; allows the user to specify I/O devices extents, and certain file attributes for use by a program at execution time. DLBL can also be used to modify or delete previously defined disk file descriptions.

Module Name	Entry Points	Function
DMSDLK	DMSDLK	Interface for the CMS/DOS DOSLKED command. Link-edit the relocatable output of the language processors. Once link-edited, these core image phases are added to the end of the specified DOSLIB.
DMSDMP	DMSDMP DMSPDP	Simulates the VSE $$BDUMP. A CP DUMP command is issued, directing the dump to a virtual printer. Simulates IDUMP, JDUMP, and PDUMP. For IDUMP, the PRINTL macro is issued. For JDUMP and PDUMP, a CP DUMP command is issued directing the dump to a virtual printer.
DMSDOS	DMSDOS	Provides DOS SVC support. Interprets DOS SVC codes and passes control to appropriate routines for execution (for example, OPEN, CLOSE, FETCH, EXCP).
DMSDRO	DMSDRO	Deletes a server from the list of servers to be notified when communications end.
DMSDRP	DMSDRP	Processes the DROP command.
DMSDSK	DMSDSK	Dumps a disk file to cards or loads files from card to disk.
DMSDSL	DMSDSL	Provides capability to delete members (phases) of a DOSLIB library; also, to compress a DOSLIB library; also, to list the members (phases) of a DOSLIB library.
DMSDSV	DMSDSV	Lists the directories of DOS private of system packs.
DMSEDC	DMSEDC	Arranges compound (overstruck) characters into an ordered form and disregards tab characters as special characters.
DMSEDF	DMSEDF	Provides the Editor with the proper settings (CASE, TAB, FORMAT, SERIAL, etc.) by filetype. Contains nonexecutable code for reference by DMSEDI.
DMSEDI	DMSEDI	Modifies the contents of an existing file or creates a new file for editing.
DMSEDX	DMSEDX	Performs initialization for the CMS Editor.
DMSEIO	DMSEIO DMSEIOI	Processes EXECIO command. Initialization routine.
DMSERD	DMSERDBF DMSEWRBF	Reads one or more items from a specified 512-, 1K-, 2K-, or 4K-byte formatted disk. Writes one or more items from a specified 512-, 1K-, 2K-, or 4K-byte formatted disk.
DMSERR	DMSERR	Builds a message to be written at the virtual console by DMSCWR.
DMSERS	DMSERS	Deletes a file or related group of files from read/write disks.
DMSETR	DMSETR	Provides SVC 98 EXTRACT macro support. Called by DMSDOS.
DMSEVC	DMSEVC	Issues and SVC 202 to an SRPI macro.
DMSEXD	DMSEXD	Processes the EXECDROP command.
DMSEXE	DMSEXE	Processes an EXEC 2 file.

Module Name	Entry Points	Function
DMSEXG	DMSEXG	Processes the DCSSGEN command.
DMSEXI	DMSEXI	Determine whether to call CMS EXEC, EXEC 2 processor, of System Product Interpreter. (DMSEXT, DMSEXE or DMSREX.)
DMSEXL	DMSEXL	Processes the EXECLOAD command.
DMSEXM	DMSEXM	Processes the EXECMAP command.
DMSEXQ	DMSEXQ	Processes the EXECSTAT command.
DMSEXT	DMSEXT	Processes a CMS EXEC file.
DMSFCH	DMSFCH	Bring a specified phase into storage from a system or private core image library or from a CMS DOSLIB library. DMSFCH is invoked via SVC 1, 2, or 4 or via the FETCH command.
DMSFET	DMSFET	Provides an interface for the FETCH command; also, provides the capability to start execution of a specified phase.
DMSFLD	DMSFLD	Interprets OS JCL DD parameters for use by CMS.
DMSFLE	DMSFLE	Processes the CLEAR and LIST functions for the FILEDEF command.
DMSFLO	DMSFLOLV DMSFLONE DMSFLODN DMSFLO18 DMSFLO9T DMSFLO7T DMSFLOTR DMSFLODC DMSFLOSY	Processes the LEAVE option of the FILEDEF command. Processes the NOEOV option of the FILEDEF command. Processes the DEN option of the FILEDEF command. Processes the 18TRACK option of the FILEDEF command. Processes the 9TRACK option of the FILEDEF command. Processes the 7TRACK option of the FILEDEF command. Processes the TRTCH option of the FILEDEF command. Performs FILEDEF density checking. Processes the SYSPARM option of the FILEDEF command.
DMSFNC	DMSFNC DMSFNCSV	Nucleus resident command name table. Standard SVC table.
DMSFND	DMSFND	Finds the message table address.
DMSFNS	DMSFNSA DMSFNSE DMSFNST	Closes one or more input or output disk files. Closes a particular file without updating the directory or removing it from the active file table. Temporarily closes all output files for a given disk.
DMSFOR	DMSFOR	Physically initializes a disk space for the CMS data management routines. For an existing disk, any information on the disk may be destroyed. The label may be changed and the number of cylinders allowed may be changed. Reads and writes one track at a time.
DMSFRC	DMSFRC	Called by DMSFRE. Cache handler for FBDs (free block descriptors) for storage management.

Module Name	Entry Points	Function
DMSFRE	DMSFREB	Called as a result of the DMSFREE and DMSFRET macro calls. Allocates or releases a block of storage depending upon the code in NUCON location CODE203.
	DMSFREES	Called as a result of the SVCFREE macro call. The size of the block is loaded from the PLIST and a DMSFREE macro is executed. Upon return, the address of the allocated block is stored into the PLIST.
	DMSFRETS	Called as a result of the SVCFRET macro call. The size and address of the block to be released are loaded from the PLIST and a DMSFRET macro is executed.
	DMSFREEX	Called as a result of a BALR to the address in the NUCON location AFREE. Executes the DMSFREE macro.
	DMSFRETX	Called as a result of a BALR to the address in the NUCON location AFRET. Executes the DMSFRET macro.
	DMSFRES	Called as a result of executing the DMSFRES macro. DMSFRES processes the following service routines: CKOFF, INIT1, INIT2, CHECKS, UREC, and CALOC.
DMSGET	DMSGET	Processes the GET command.
DMSGIO	DMSGIO	Creates the DIAGNOSE and CCWs for an I/O operation to a display terminal from a virtual machine.
DMSGLB	DMSGLB	Defines the macro libraries to be searched during assembler processing. Defines text libraries to be searched by the loader for any unresolved external references. Defines DOS libraries to be searched by the DOS fetch for a requested phase load. Defines the load libraries to be searched by the OS fetch support for a requested OS simulated load module.
DMSGLO	DMSGLO	Handles GLOBALV command requests to: 1) define global variables for short term use in table(s) in storage, or long term use in CMS files; 2) retrieve and stack variables for use by EXECs.
	DMSGLOIN	Gets and initializes GLOBALV workarea.
DMSGMF	DMSGMF	Provides support for SVC 107 GETFLD and MODFLD macros. Called by DMSDOS.
DMSGND	DMSGND	Generates auxiliary system status table.
DMSGRN	DMSGRN	Edits STAGE1 output (STAGE2 input), builds 3705 assembler files, link-edits text files and an EXEC macro file.
DMSGRQ	DMSGRQ	Gets service request information from the CPRB and sets the appropriate SRPI variables in the server.
DMSGTM	DMSGTM	Provides support for SVC 34 GETIME macro. Called by DMSDOS.
DMSGTU	DMSGTU	Issues a SUBCOM SRPI query. Checks to see if the user word is set up to point to the Global Control Block and returns the SCBLOCK address.
DMSGVE	DMSGVE	Provides support for SVC 99 GETVCE macro. Called by DMSDOS and DMSGMF.

Module Name	Entry Points	Function
DMSHDI	DMSHDI (HNDINT)	Sets the CMS interruption handling functions to transfer control to a given location for an I/O device other than those normally handled by CMS, or clears previously initialized I/O interruption handling.
DMSHDS	DMSHDS	Initializes the SVCINT SVC interruption handler to transfer control to a given location for a specific SVC number (other than 202) or to clear such previous handling.
DMSHID	DMSHID	Processes the HIDE command.
DMSHLB	DMSHLB	Processes and builds output for .BX HELP format word for the HELPCONV command.
DMSHLC	DMSHLC DMSHLCOC DMSHLCCK	Processes options from a non-tokenized string of words. Processes options keywords. Determines if a line matches the searched abbreviation.
DMSHLD	DMSHLD	HELPCONV facility communication module, loaded into free storage by DMSHLI.
DMSHLI	DMSHLI	Contains HELPCONV facility initialization routines.
DMSHLL	DMSHLL	Performs HELP initialization function and handles TTY-type output.
DMSHLP	DMSHLP	HELPCONV facility module for processing HELP description file input.
DMSHLR	DMSHLR DMSHLRCL	Processes HELP subcommand calls from HELPXED. Closes a HELP file.
DMSHLS	DMSHLS SWRTPREP SWRTIO GOPEN GREAD GCLOSE	Contains HELPCONV facility I/O routines. Determines virtual terminal characteristics and acquires buffer storage. Performs normal virtual terminal I/O. Performs OPEN function for HELP description. Routine to read HELP file input. Routine to perform file closing functions on exit.
DMSHLT	DMSHLT	Searches for the proper HELP file.
DMSHLZ	DMSHLZ DMSHLZOP DMSHLZQU DMSHLZFD DMSHLZCS	Replaces pairs of highlighting characters with a single blank. Processes options from the parser PLIST. Queries the userid. Determines if a HELP disk is accessed. Determines the validity of a .cs control word.
DMSHTB	DMSHTB	Builds a hyperblock mapping table for R/O disks, and builds hash tables for R/W disks.
DMSIAC	DMSIAC	Validates the subcommand and checks the prefix. Invokes appropriate routine to handle subcommand.
DMSIDE	IDENTIFY	Display or stack information about the virtual machine.
DMSIMA	DMSIMAMD	Implements the IMAGEMOD command. This command is used to modify specific members of a 3800 named system. With this command, you can dynamically delete, add, replace, and generate members for a named system.
DMSIMM	DMSIMM	Handles the IMMCMD macro and the IMMCMD command.

Module Name	Entry Points	Function
DMSINA	DMSINA	Handles either user-defined synonyms or abbreviations or system-defined synonyms for command names.
DMSIND	DMSINDI DMSINDS	Performs S- and Y-STAT initialization. Validates the S- and Y-STAT.
DMSINI	DMSINIR DMSINIW	Reads a nucleus into main storage. Writes a nucleus onto a DASD unit.
DMSINM	DMSINM (GETCLK) (CMSTIMER)	Obtains the time from the CP timer.
DMSINQ	DMSINQ	Performs message initialization for DMSINI and DMSINQ.
DMSINS	DMSINS	Controls initialization of the CMS nucleus.
DMSINT	DMSINT DMSINTAB SUBSET	Reads CMS commands from the terminal and executes them. Entry is from DMSINS. Entry from DMSABN. CMS subset entry.
DMSINV	DMSINV	Performs Vector Facility initialization.
DMSIOW	DMSIOW, WAIT, DMSIOWR, WAITRTN	Places the virtual CPU in the wait state until the completion of an I/O operation on one or more devices.
DMSITE	DMSITE, EXTINT, DMSITET, TRAP	Processes external interruptions.
DMSITI	DMSITI, IOINT	This module is entered when an I/O operation causes the I/O new PSW to be loaded. This module handles all I/O interruptions, passes control to the interruption processing routine, and returns control to the interrupted program.
DMSITP	DMSITP	Processes program interruptions and processes SPIE exits.
DMSITS	DMSITS DMSITS1 DMSITSCR DMSITSNU DMSITSOR DMSITSK DMSITSXS DMSITSR DMSITSSB	Avoids CP overhead due to SVC call. Address pointed to by the CMS SVC new PSW. This point is entered whenever an SVC interruption occurs. Return point to which a program called by a CMS SVC returns when it is finished processing. NUCEXT handling. Return point to which a program called by an OS SVC returns when it is finished processing. Called by an SVC by the DMSKEY macro. Called by an SVC from the DMSEXS macro. This is the DMSITS recovery and reinitialization routine, called by DMSABN. DMSABN is the ABEND recovery routine. SUBCOM handling.
DMSIUC	DMSIUC	Handles the CMSIUCV and HNDIUCV macros.
DMSLAB	DMSLAB	Simulates the VSE LABEL macro.

Module Name	Entry Points	Function
DMSLAD	DMSLAD, ADTLKP DMSLADN, ADTNXT DMSLADR DMSLADW DMSLADAD	Finds the active disk table block whose mode matches the one supplied by the caller. Finds the first or the next ADT block in the active disk table. Releases an active disk table block. Finds the read or write disk according to input parameters. Modifies the file status table chain to include an auxiliary directory, or clears the auxiliary directory from the chain.
DMSLAF	DMSLAF, ACTLKP DMSLAFNX, ACTNXT DMSLAFFE, ACTFREE DMSLAFFT, ACTFRET	Finds the active file table block whose filename, filetype and filemode match the one supplied by the caller. Finds the next or first AFT block in the active file table. Finds an empty block in the active file table or adds a new block from free storage to the active file table, if necessary, and places a file status entry (if given) into the AFT block. Removes an AFT block from the active file table and returns it to free storage if necessary.
DMSLBD	DMSLBD	Allows the user to specify tape label information that will be used by a program at execution time (the parameters are similar to those of the DOS TLBL statement or the tape options of the OS data definition statement). LABELDEF can also be used to modify, delete, and list previously described label descriptions.
DMSLBM	DMSLBM	Generates a macro library, adds macros to an existing library, and lists the dictionary of an existing macro library.
DMSLBR	DMSLBR	Simulates the VSE LBROPEN, LBRFIND, and LBRGET macros as required by the VSE ESERV utility program.
DMSLBT	DMSLBT, TXTLIB	Creates a text library, adds text files to an existing text library, creates a disk file that lists the control section and entry point names in a text library or types, at the terminal, the control section and entry point names in a text library.
DMSLCK	DMSLCK	SVC 110 LOCK/UNLOCK macro support. Called by DMSDOS.
DMSLDF	DMSLDF	Provides support for SVCs 1, 2, 4, and 65 that correspond to macros FETCH, FETCH, LOAD, and CDLOAD, respectively. Called by DMSDOS.

Module Name	Entry Points	Function
DMSLDR	DMSLDRA	Begins execution of a group of programs loaded into real storage. Definition of all undefined programs is established at location zero. Entered from the START command or internally from DMSLDRB LDT routine if START is specified.
	DMSLDRB	Processes TEXT files that may contain the following cards: SLC, ICS, ESD, TXT, REP, RLD, END, LDT, LIBRARY, and ENTRY. Entered from DMSLDP when the load function is requested.
	DMSLDRC	Does the processing required by various loader routines when an invalid card is detected in a text file.
	DMSLDRD	Does the processing required when a fatal I/O error is detected in a text file.
DMSLDS	DMSLDS	Lists information about specified data sets residing on an OS disk. Processes the LISTDS command.
DMSLFS	DMSLFS, TYPSRCH	Finds a specified FST entry within the FST blocks for read-only or read/write disks.
	DMSLFSHM	Updates the R/W disk hash tables for the fileid hashing mechanism.
DMSLGT	DMSLGTA	Entered from DMSLDRB if not a dynamic load. Frees all the TXTLIB blocks on the TXTLIB chain.
	DMSLGTB	Reads TXTLIB directories into a chain of free storage directory blocks. Entered from DMSLDRB.
DMSLIB	DMSLIB	Searches TEXT libraries for undefined symbols and closes the libraries.
DMSLIC	DMSLIC	Provides support for SVC 50 LIOCS ERROR. Called by DMSDOS.
DMSLIO	DMSLIO	Creates the load map on disk and types it at the terminal. Performs disk and typewriter output for DMSLDR.
DMSLKD	DMSLKD	Provides an interface between CMS and the VS1 linkage editor.
DMSLLU	DMSLLU	Lists the assignments of logical units.
DMSLMX	DMSLMX	Establishes selected CMS commands (e.g., TAPE) as nucleus extensions.
DMSLOA	DMSLOA	Processes the LOAD and INCLUDE commands to invoke the relocating loader.
DMSLOS	DMSLOS	Provides load and relocate support for OS load modules and CMS LOADLIB modules.
DMSLSB	DMSLSBA	Hexadecimal to binary conversion routine.
	DMSLSBB	Adds a symbol to the string of locations waiting for an undefined symbol to be defined.
	DMSLBC	Removes the undefined bit from the REFTBL entry and replaces the ADCON with the relocated value.
	DMSLBD	Processes LDR options.
DMSLST	DMSLSTA	Processes the LISTFILE command. Prints information about the specified files.

Module Name	Entry Points	Function
DMSLSY	DMSLSY	Generates a unique character string of the form Z000001 for private code symbols.
DMSMAX	DMSMAX	Processes the MAXIMIZE command.
DMSMCM	DMSMCM	Provides support for SVC 5 MVCOM macro. Called by DMSDOS.
DMSMDP	DMSMSP	Types the load map associated with the specified file on the terminal.
DMSMGM	DMSMGM	Processes message requests from the DMSMSG and APPLMSG macros and the XMITMSG command.
DMSMIN	DMSMIN	Processes the MINIMIZE command.
DMSMKS	DMSMKS	Creates a CPRB stack block and adds it to the chain of stack blocks.
DMSMOD	GENMOD	Processes the GENMOD command to create a file that is a core image copy of the loaded object code.
	LOADMOD	Processes the LOADMOD command to load a file that is in core image form.
DMSMVE	DMSMVE	Transfers data between two specified OS ddnames, the ddnames may specify any devices or disk files supported by the CMS system.
DMSMVG	DMSMVG	Handles input for the MOVEFILE command when the input is a DOS FBA file.
DMSNAM	DMSNAM	Processes the NAMEFIND command. Displays or stacks information contained in a 'NAMES' file.
	DMSNAMI	Installs NAMEFIND as a nucleus extension.
DMSNCP	DMSNCP	Reads a 3705 control program module (Emulator Program or Network Control Program) in OS load module format and writes a page-format core image copy on a VM/SP system volume.
DMSNUC	DMSNUC	Contains CSECTS for nucleus work areas and permanent storage.
	NUCON	Nucleus constant area.
	SYSREF	Nucleus address table.
	DEVTAB	Device table.
	ADTSECT	Active disk table.
	AFTSECT	Active file table.
	EXTSECT	External interruption storage.
	IOSECT	I/O interruption storage.
	PGMSECT	Program Interruption storage.
	SVCSECT	SVC interruption storage.
	DIOSECT	Disk I/O storage.
	FVS	File system storage.
	OPSECT	Parameter lists.
	CVTSECT	Simulated OS CVT.
	DBGSECT	Debug storage.
	TSOBLKS	TSO control blocks.
DMSNXD	DMSNXD	Processes the NUCXDROP command.
DMSNXL	DMSNXL	Processes the NUCXLOAD command.
DMSNXM	DMSNXM	Processes the NUCXMAP command.

Module Name	Entry Points	Function
DMSOME	DMSOME	Marks the end of the CMS nucleus.
DMSOPL	DMSOPL	Reads the appropriate system directory records and headers and determines if the specified libraries contain any active members. Returns the disk address of the specified system library and indicates whether or not there are active members to be accessed on the disk.
DMSOPT	DMSOPT	Sets VSE options in the System Communications Region as specified by the OPTION command.
DMSOR1	DMSOR1	Relocates all DTF (define the file) Table address constants to executable storage addresses. (Called by $$BOPENR via SVC 2).
DMSOR2	DMSOR2	Relocates all DTF (define the file) Table address constants to executable storage addresses. (Called by DMSOR1).
DMSOR3	DMSOR3	Relocates all DTF (define the file) Table address constants to executable storage addresses. (Called by DMSOR2).
DMSOSR	DMSOSR	Allows user to invoke a program from a CMS LOADLIB or an OS module library.
DMSOVR	DMSOVR	Analyzes the SVCTRACE command parameter list and loads the DMSOVS tracing routine.
DMSOVS	DMSOVS	Provides trace information requested by the SVCTRACE command.
DMSPAG		Provides CMS page management.
	DMSPAGRL	Receives notification of pages eligible for release.
	DMSPAGGT	Receives notification of pages not eligible for release.
	DMSPAGFL	Releases all eligible pages if SET RELPAGE ON is in effect.
DMSPAR	DMSPAR	Parsing facility driver. Called as a result of the PARSECMD macro.
DMSPBK	DMSPBK	Creates SRPI variable names as a concatenation of prefix and base names.
	DMSPBKGL	Gets a truncated string from EXECCOMM.
DMSPCA	DMSPCA	Convert commands, verification phase.
DMSPCB	DMSPCB	Convert commands, conversion phase.
DMSPCC	DMSPCC	Convert commands, driver.
DMSPCL	DMSPCLEM	Parsing facility, syntax error messages.
	DMSPCLMD	Parsing facility, parse modifier clause of parameter list.
	DMSPCLOR	Parsing facility, parse operand clause of parameter list.
	DMSPCLOT	Parsing facility, parse option clause of parameter list.
DMSPCR	DMSPCR	Convert commands, read and syntax check DLCS input file.
DMSPCT	DMSPCT	Convert commands, write command name translation table.
DMSPCW	DMSPCW	Convert commands, write output syntax definition table.

Module Name	Entry Points	Function
DMSPDB	DMSPDB	Manipulate double byte character strings.
DMSPIO	DMSPIO DMSPIOCC DMSPIOSI	Prints one line. Puts CCWs to select translate table (for virtual 3800) and to print the data, plus the data itself, in the caller's buffer. Prints the caller's buffer, issues an SIO to the virtual printer, and analyzes the resulting status.
DMSPKR	DMSPKR	Command name translation and synonym lookup driver.
DMSPKT	DMSPKT	Translate command names.
DMSPMD	DMSPMD	Parsing facility exec interface (PARSECMD command).
DMSPNT	DMSPNT	Places the address of a file status table entry in the active file table (if necessary), and sets the read pointer or write pointer for that file to a given item number within the file.
DMSPOA	DMSPOA	Contains the action handling routines for the programmable operator facility.
DMSPOC	DMSPOC	Contains the console handler routines for the programmable operator facility.
DMSPOD	DMSPOD	Contains data shared by the rest of the modules comprising the programmable operator mainline and also messages issued by the programmable operator mainline.
DMSPOE	DMSPOE	Contains exit routines for the programmable operator facility.
DMSPOL	DMSPOL	Contains the supplied action routine for loading a routing table for the programmable operator facility.
DMSPON	DMSPON	Contains node-checking support routines for the programmable operator facility.
DMSPOP	DMSPOP	Establishes communication with the PMX, and handles message passing, programmable operator initialization, and termination.
DMSPOQ	DMSPOQ	Contains the programmable operator subroutines for checking RTABLEs.
DMSPOR	DMSPOR	Contains the supplied action routines for the programmable operator.
DMSPOS	DMSPOS	Contains the supplied action routine for routing a message for the programmable operator facility.
DMSPPL	DMSPPL	Parsing facility, produce parsed parameter lists.
DMSPPO	DMSPPO	Processes the POP command.
DMSPRB	 DMSPRI DMSPRJ	Parsing facility, routine parsing driver module. Command routine parsing A. Command routine parsing B.
DMSPRE	DMSPREEP	Combine, link, and relocate multiple text (object) files into a single text file.
DMSPRT	DMSPRT	Prints CMS files.
DMSPRV	DMSPRV	Copies procedures from the VSE system procedure library to a specified output device.

Module Name	Entry Points	Function
DMSPSC	DMSPSCAN DMSPSCTK	Parsing facility, scan and identify tokens. Parsing facility, scan tokens from input parameter list.
DMSPSM	DMSPSMGT DMSPSMEX	Parsing facility storage manager, get storage for output parameter lists. Parsing facility storage manager, expand storage for output parameter lists.
DMSPST	DMSPST	Processes the POSITION command.
DMSPTC	DMSPTC	Parsing facility, find command syntax definition in syntax definition table.
DMSPTK	DMSPTK	Parsing facility, find keyword translation in syntax definition table.
DMSPTL	DMSPTL	Parsing facility, find modifier definition in syntax definition table.
DMSPTR	DMSPTR	Parsing facility, find operand definition in syntax definition table.
DMSPTT	DMSPTT	Parsing facility, find option definition in syntax definition table.
DMSPUN	DMSPUN	Punches CMS files to the virtual card punch.
DMSPUT	DMSPUT	Processes the PUT command.
DMSPVF	DMSPVF	Parsing facility, validate all non-keyword data.
DMSQRF	DMSQRF	Processes the QUERY commands for windowing functions.
DMSQRS	DMSQRS	Processes the QUERY DISK and SEARCH subcommands from DMSQRY.
DMSQRT	DMSQRT	Processes the following QUERY subcommands: ABBREV, AUTOREAD, BLIP, CMSTYPE, EXECTRAC, HASH, IMESCAPE, IMPCP, IMPEX, INSTSEG, LDRTBLS, PROTECT, RDYMSG, RELPAGE, SYSNAMES, and CMSLEVEL.
DMSQRU	DMSQRU	Processes the QUERY FILEDEF and LABELDEF subcommands.
DMSQRV	DMSQRV	Processes the QUERY INPUT, OUTPUT, and SYNONYM subcommands.
DMSQRW	DMSQRW	Processes the QUERY LIBRARY, MACLIB, TXTLIB, DOSLIB, and LOADLIB subcommands.
DMSQRX	DMSQRX	Processes the QUERY DOSPART, OPTION, DOSLNCNT, UPSI, DLBL, and DOS subcommands.
DMSQRY	DMSQRYI DMSQRY	Loads QUERY as a nucleus extension. Initializes QUERY work area, if necessary. Processes the QUERY command by passing control to one of the following: 1. Another QUERY module: for a CMS subcommand with the correct syntax. 2. CP: for a subcommand, other than a CMS subcommand. 3. Caller: for a syntax error.

Module Name	Entry Points	Function
DMSQRZ	DMSQRZ	Non-executable constants used by DMSQRY to initialize the work area for the CMS QUERY command.
DMSRCN	DMSRCN	Performs character conversion, console I/O, general services, and all arithmetic.
DMSRDC	READCARD	Reads cards and assigns the indicated filename.
DMSRDR	DMSRDR	Processes the RDR command. Stacks and displays the characteristics of the first file in the virtual reader.
DMSREF	DMSREF	Processes the REFRESH command.
DMSRES	DMSRES	Processes the RESTORE command.
DMSREV	DMSREV	Evaluates expressions.
DMSREX	DMSREX DMSREXVE DMSREXMS	Reads the EXEC file; calls DMSRIN. Handles the EXECCOMM interface. Retrieves the text of error messages.
DMSRFN	DMSRFN DMSRFNTB	Performs built-in functions, invokes external functions. Table of built-in functions.
DMSRIN	DMSRIN DMSRINIF DMSRINCK	Parses input data, controls most execution decisions, and passes clauses to DMSRXE for execution. Entry point for recursive call with INTERPRET, subroutine/function calls, and label searches. Table of valid keywords.
DMSRLD	DMSRLD	Called by DMSNXL (NUCXLOAD). Loads CMS modules and relocates the address constants of CMS load modules.
DMSRNE	DMSRNE	Provides an interface for the CMS Editor RENUM subcommand, which renumbers files with filetypes of VSBASIC and FREEFORT.
DMSRNM	DMSRNM	Processes the RENAME command. Changes the fileid of the specified file.
DMSROU	DMSROU	Processes the ROUTE command.
DMSROS	DMSROS ROSACC DMSROS+4 ROSSTT DMSROS+8 ROSRPS DMSROS+12 ROSFIND DMSROS+16 ROSNTPTB	Accesses OS disks. Verifies the existence of OS disks. Reads OS disks. Finds a member in an OS partitioned data set. Performs NOTE, POINT, and BSP functions.
DMSRPG	DMSRPG	Provides support for SVC 85 RELPAG macro. Called by DMSDOS.
DMSRRV	DMSRRV	Provides the capability to copy (to an output device) modules residing on DOS system or private relocatable libraries.

Module Name	Entry Points	Function
DMSRSF	DMSRSF	Loads RXSYSFN as a nucleus extension.
	DMSSYSFN	Processes function load requests.
	RXDIAG	Processes the DIAG function.
	RXDIAGRC	Processes the DIAGRC function.
	RXCMSFLA	Processes the CMSFLAG function.
	RXSTORAG	Processes the STORAGE function.
DMSRSV	DMSRSV	Distributes all the blocks of a mini-disk between the directory file, allocation map file, and user's file. DMSRSV sets up pointer blocks for each of these files.
DMSRTC	DMSRTC	Formats and displays trace information.
DMSRTE	DMSRTE	Calls: DMSCST to add the CPRB pointer to the stack. DMSSRH to find the server and invoke it. DMSUST to remove the CPRB pointer from the stack. DMSVLD to validate the copied CPRB.
DMSRTV	DMSRTV	Retrieves messages from the message table.
DMSRXE	DMSRXE	Executes individual clauses.
DMSRVA	DMSRVA	Accesses and maintains System Product Interpreter variables.
DMSSAB	DMSSAB	Processes OS ABEND macros.
DMSSBD	DMSSBD	Accesses data set records directly by item number. It converts record identifications given by OS BDAM macros into item numbers and uses these item numbers to access records.
DMSSBS		Process OS BSAM READ and WRITE macros.
	DMSSBSRT	Entry for error return from call to DMSSBD.
DMSSCL	DMSSCL	Processes the SCROLL command.
DMSSCN	DMSSCN	Transforms the input line from a series of arguments to a series parameter strings.
DMSSCR	DMSSCR	Loads display buffers and issues a macro resulting in a CP DIAGNOSE to write to the display terminal.
DMSSCT	DMSSCTNP	Process OS POINT, NOTE, CHECK, and FIND (type C) macros.
	DMSSCTCK	Process OS CHECK macro.
	DMSSCTCE	Handles QSAM I/O errors for DMSSQS and PDS and keys errors for DMSSOP.
	DMSSCTTP	SETs and CLEARs the CMS tape end-of-volume exits.
DMSSEB	DMSSEB	Calls device I/O routines to do I/O and sets up ECB and IOB return codes.
DMSSEF	DMSSEF	Processes the SET commands for windowing functions.
DMSSET	DMSSET	Processes the following SET subcommands: BLIP, RDYMSG, LDRTBLS, RELPAGE, INPUT, OUTPUT, ABBREV, REDTYPE, IMPEX, IMPCP, PROTECT, AUTOREAD, SYSNAME, NONSHARE, CMSTYPE, IMESCAPE, INSTSEG, EXECTRAC, DOS, DOSLNCNT, UPSI, and DOSPART.
DMSSFD	DMSSFD	Saves file status tables and the related information into a DCSS so that users can share file directory information for a R/O EDF disk.

Module Name	Entry Points	Function
DMSSFF	DMSSFF	Provides overlay support for OS load modules.
DMSSHO	DMSSHO	Processes the SHOW command.
DMSSIG	DMSSIG	The anchor table for the SSTAT and YSTAT. Also marks the end of the CMS nucleus code.
DMSSLG		Processes the SET LANGUAGE command.
	DMSSLGAL	Copies the LANGBLK pointed to in the PLIST and adds it to the chain of LANGBLKs anchored by NUCLANGA (if one for that application is not already on the chain). CMS uses the SVC name LANGADD to do this.
	DMSSLGSV	Returns the address of a language control block (LANGBLK) on the chain of LANGBLKs anchored by NUCLANGA. CMS uses the SVC function LANGFIND to do this search.
DMSSMG	DMSMSG	Interface to DMSFREE and DMSFRET.
DMSSIZ	DMSSIZ	Processes the SIZE command.
DMSSLN	DMSSLN	Handles OS contents management requests issued under CMS (LINK, LOAD, XCTL, DELETE, ATTACH, EXIT).
DMSSMN	DMSSMN	Processes OS FREEMAIN and GETMAIN macros and CMS calls DMSSMNSB and DMSSMNST.
DMSSOP	DMSSOP	Processes OS OPEN and CLOSE macros.
DMSSPM	DMSSPM	Determines the location in storage and the length of a SYSPARM string.
DMSSPR	DMSSPR	Processes the SETPRT command. This command sets up a virtual 3800 printer spool file for a CMS user. With the SETPRT command, a user can select the character arrangement tables, copy modification modules, FCB, and forms overlay frame for printing files with a virtual 3800.
DMSSQS	DMSSQS	Analyzes record formats and sets up the buffers for GET, PUT, and PUTX requests.
DMSSRE	DMSSRE	Main entry for the SRPI subcommand environment. Calls DMSGTU to get pointer to SCBLOCK and if the server is an SRPI macro passes control to DMSIAC; otherwise, passes control to DMSRTE.
DMSSRH	DMSSRH	Searches through the SRPI search sequence (nucleus extension, SRPI macros, modules) to find the server and then it passes control to the server found.
DMSSRP	DMSSRP	Gets service reply information fron server SRPI variables and updates the CPRB.
DMSSRQ	DMSSRQ	Creates new CPRB for request. Gets service information from requester SRPI variables and updates the CPRB. Calls DMSRTE to invoke server. Gets service reply information from the CPRB and sets the appropriate SRPI variables in the requester. Destroys old CPRB.
DMSSRT	DMSSRT	Arranges records within a file in descending sequential order.

Module Name	Entry Points	Function
DMSSRV	DMSSRV	Provides capability to copy books from a system or private source statement library to a specified output device.
DMSSSK	DMSSSK	Sets storage protect key for a specified saved system.
DMSSTC	DMSSTC	Saves the pointer to the copied CPRB in the CPRB stack.
DMSSTG		Processes CMS calls to DMSSTGST and DMSSTGSB (STRINIT) and storage service routines.
	DMSSTGOS	Processes the EXECOS command.
	DMSSTGSB	STRINIT.
	DMSSTGST	STRINIT.
	DMSSTGCL	OS exit reset routine.
	DMSSTGSV	Service routine to change nucleus variables.
	DMSSTGAT	Initializes storage and sets up an anchor table.
DMSSTP	DMSSTP	Calls tape I/O routines to do I/O and sets up ECB and IOB return codes.
DMSSTT	DMSSTT	Locates the file status table entry for a given file and, if found, provides the caller with the address of the entry.
	DMSSTTV	Processes the VALIDATE command.
DMSSTX	DMSSTX	Provides support for SVCs 16, 17, 37, and 95 that correspond to macros STXIT PC, EXIT PC, STXIT AB, and EXIT AB, respectively. Called by DMSDOS.
DMSSUB	DMSSUB	SVC 105 SUBSID macro support. Called by DMSDOS.
DMSSVL	DMSSVL	SVC 75 SECTVAL macro support. Called by DMSDOS.
DMSSVN	DMSSVN	Processes the OS WAIT and POST macros.
DMSSVQ	DMSSVQGB	Obtains a workarea for a CMS module from a stack of USER storage.
	DMSSVQFB	Releases a workarea for a CMS module from a stack of USER storage.
	DMSSVQGD	Obtains a workarea for a CMS module from CMS USER storage.
	DMSSVQFD	Releases a workarea for a CMS module from CMS USER storage.
	DMSSVQPB	Reinitializes a stack of CMS USER storage.
DMSSVT	DMSSVT	FIND, STOW, DEVTYPE, TRKBAL, WTO, WTOR, EXTRACT, IDENTIFY, CHAP, TTIMER, STIMER, DEQ, SNAP, ENQ, FREEDBUF, STAE, DETACH, CHKPT, RDJFCB, SYNAD, BACKSPACE, and STAX.
DMSSVU	DMSSVU	Builds a keys file when a data file using keys is opened and saves the keys in the data file when it is closed.
DMSSYN	SYNONYM	Processes the SYNONYM command. Sets up user-defined command names and abbreviations for CMS commands.
DMSTCA	DMSTCA	Provides CMS support for issuing a DIAGNOSE code X'70' to receive time-of-day clock accounting information.
DMSTIO	DMSTIO	Reads or writes a tape record or controls tape positioning.

Module Name	Entry Points	Function
DMSTLB	DMSTLB	Processes IBM standard tape labels for OS simulation, CMS/DOS, CMS commands, and TAPESL macro. Also provides linkage to nonstandard user label routines for OS simulation and CMS commands.
DMSTMA	DMSTMA	Reads and IEHMOVE unloaded PDS from tape and places it in a CMS MACLIB.
DMSTPD	DMSTPD	Reads a tape consisting of card image members of a PDS and creates CMS disk files for each member of the data set. The PDS option allows reading unblocked tapes produced by the OS IEBPTPCH utility or blocked tapes produced by the OS IEHMOVE utility. The UPDATE option provides the "./ ADD" function to blocked or unblocked tapes produced by the IEBUPDTE utility.
DMSTPE	DMSTPE	Control DVOL1 and WVOL1 functions of the TAPE command. Control functions include REW, RUN, WTM, ERG, BSR, BSF, FSR, FSF.
DMSTPF	DMSTPF	Load, scan, and skip functions of the TAPE command.
DMSTPG	DMSTPG	Dump functions of TAPE command.
DMSTPH	DMSTPH	Initialization and parameter list validation for the TAPE command.
DMSTPI	DMSTPI	The DMSTPI text file is the driver for the TAPE command which performs certain tape functions, such as: dump a CMS file, set tape mode, display or write VOL1 labels, scan, skip, rewind, RUN, FSF, FSR, BSF, BSR, ERG, and WTM. The DMSTPI module is composed of the DMSTPI, DMSTPJ, DMSTPE, DMSTPF, DMSTPH, and DMSTPG text files. DMSTPI is established as a nucleus extension by the DMSLMX bootstrap module.
DMSTQQ	DMSTQQ DMSTQQX	Allocates a 200-byte first chain link (FCL) to a calling program. Makes a 200-byte disk area no longer needed by one program available for allocation to another program.
DMSTRK	DMSTRKA DMSSTRKX	Allocates an 800-byte disk area to a calling program. Makes an 800-byte disk area that is not longer needed by one program available for allocation to another.
DMSTVS	DMSTVS	Provides tape volume switching for OS simulation.
DMSTYP	TYPE	Processes the TYPE command. Types all or a specified part of a given file on the user's console.
DMSUPD	DMSUPD	Processes the UPDATE command. Updates source files according to specifications in update files. Multiple updates can be made, according to specifications in control files that designate the update files.
DMSUPP	DMSUPPER	Converts data from lowercase to uppercase.
DMSUSR	DMSUSR	Generates second level request.
DMSUST	DMSUST	Returns the top most pointers in the CPRB stack to both the current and the copied CPRB then removes them from the CPRB stack.
DMSUTL	DMSUTL	List, copy, or compress LOADLIBs.

Module Name	Entry Points	Function
DMSVAN	DMSVAN	First table of Access Method Services nonshared (nonreentrant) modules.
DMSVAS	DMSVAS	Contains a table of Access Method Services shared (reentrant) modules.
DMSVAX	DMSVAX	Second table of Access Method Services nonshared (nonreentrant) modules.
DMSVBM	DMSVBM	Contains table of simulated VSE phases located in CMSBAM DCSS.
DMSVIB	DMSVIB	Loads the CMS/VSAM saved system and pass control to the CMS/VSAM interface routine, DMSVIP.
DMSVIP	DMSVIP	Finds the CMS/DOS discontiguous shared segment (DCSS); issues all necessary VSE ASSGN statements for OS user; maps all OS VSAM macro requests to VSE specifications; equivalents, where necessary; traps all transfers of control between VSAM and the OS user and sets the appropriate operating environment flags.
DSMVIS	DMSVIS	Provide support for SVC 61 GETVIS macro and for SVC 62 FREEVIS macro. Called by DMSDOS and DMSLDF.
DMSVLD	DMSVLD	Validates the CPRB returned by the server.
DMSVLT	DMSVLT	Simulates VSE $$BOSVLT transient. Provides return linkage from SAM OPEN/CLOSE routines to CMS/DOS routines.
DMSVSR	DMSVSR	Resets any flags or fields set by VSAM processing; purges the VSAM discontiguous shared segment.
DMSVVN	DMSVVN	Contains table of VSE/VSAM nonshared (nonreentrant) modules.
DMSVVS	DMSVVS	Contains table of VSE/VSAM shared (reentrant) modules.
DMSWAT	DMSWAT	Processes the WAITT command.
DMSWEX	DMSWEX	Windowing exit to handle asynchronous interrupts.
DMSWID	DMSWID DMSWIDCR	Builds a character mode data stream. Called by DMSWIW. Determines placement of the cursor on the physical screen. Called by DMSWID and DMSWIF.
DMSWIF	DMSWIF	Builds a field mode data stream. Called by DMSWIW.
DMSWIM	DMSWIM DMSWIMLT	Builds an image of the physical screen by moving data from the virtual screen buffers to the image buffers. Called by DMSWIW. Determines if a line in a window is a top reserved line, bottom reserved line, data line, pad line, or a border. Called by DMSWIM and DMSWIR.

Module Name	Entry Points	Function
DMSWIN	DMSWINCB	Performs initialization functions for windowing.
	DMSWINX	Special entry for XEDIT. Performs initialization functions for windowing.
	DMSWINRE	Reinitializes fullscreen CMS in the event of a reconnect onto a different terminal type.
	DMSWINFS	Implements the SET FULLSCREEN ON command. Defines default virtual screens, windows, and CMS PF keys.
DMSWIO	DMSWIO	Interface to call the console service to write and read the screen.
	DMSWIOWR	Sets up the call to the console I/O function to write a screen of data in fullscreen mode. Called by DMSWIW.
	DMSWIORD	Sets up the call to the console I/O function to read the physical screen. Called by DMSWIR.
	DMSWIOAL	Calls console to ring the alarm.
DMSWIR	DMSWIR	Reads the physical screen and processes the modified fields. Called by DMSWVTIT (virtual screen WAIT) and by DMSWEX.
	DMSWIRST	Gets temporary work buffer. Called by DMSWIR and DMSWIM.
	DMSWIRMF	Finds the modified fields for a virtual screen. Called by DMSWIR and DMSWVCRY.
DMSWIT	DMSWITQU	Implements the SET FULLSCREEN SUSPEND command. Severs the connection to IUCV.
	DMSWITFS	Implements the SET FULLSCREEN OFF command. Deletes default virtual screens, windows, and CMS PF keys.
DMSWIW	DMSWIW	Refreshes the physical screen. Called by DMSWVTIT (virtual screen WAIT) and by DMSREF (REFRESH command).
DMSWLR	DMSWLR	Reads console input when FULLSCREEN in ON. Provides the function for the LINERD macro.
DMSWLW	DMSWLW	Displays a line to the console when FULLSCREEN is ON. Provides the function for the LINEWRT macro.
DMSWMI		"Lowest level" windowing module. Entry points in DMSWMM and DMSWMO call corresponding entry points in DMSWMI.
	DMSWMIFD	Finds a window.
	DMSWMIGF	Gets the first window.
	DMSWMIGL	Gets the last window.
	DMSWMIGN	Gets the next window.
	DMSWMIGP	Gets the previous window.
	DMSWMIGT	Gets the top window.
	DMSWMIDF	Defines a window.
	DMSWMIDL	Deletes a window.
	DMSWMIPD	Moves a window down in the display order.
	DMSWMIMF	Moves a window to the bottom of the display order.
	DMSWMIML	Moves a window to the top of the display order.
	DMSWMIPU	Moves a window up in the display order.
	DMSWMIUM	Updates a window.

Module Name	Entry Points	Function
DMSWMM		"High level" windowing module. It performs "window mapping functions." Entry points in DMSWMM call corresponding entry points in DMSWMI.
	DMSWMMAJ	Adjusts windows during reconnect processing.
	DMSWMMCL	Clears a window.
	DMSWMMHD	Hides a window.
	DMSWMMSH	Shows a window.
	DMSWMMUX	Updates physical screen window index buffer.
	DMSWMMBA	Scrolls backward.
	DMSWMMBT	Scrolls bottom.
	DMSWMMDN	Scrolls down.
	DMSWMMFO	Scrolls forward.
	DMSWMMLE	Scrolls left.
	DMSWMMRI	Scrolls right.
	DMSWMMTP	Scrolls top.
	DMSWMMUP	Scrolls up.
DMSWMO		"High level" windowing module. It performs "window mapping functions" not performed in DMSWMM. Entry points in DMSWMO calls corresponding entry points in DMSWMI.
	DMSWMODF	Defines a window.
	DMSWMODL	Deletes a window.
	DMSWMODT	Disconnects windows.
	DMSWMOMX	Maximizes a window.
	DMSWMOMN	Minimizes a window.
	DMSWMOMV	Moves a window to a new location on the physical screen.
	DMSWMOPD	Moves a window down in the display order.
	DMSWMOMF	Moves a window to the bottom of the display order.
	DMSWMOML	Moves a window to the top of the display order.
	DMSWMOPA	Pops the topmost window connected to a specified virtual screen.
	DMSWMOPU	Moves a window up in the display order.
	DMSWMORE	Restores a window.
	DMSWMOSB	Sets window borders.
	DMSWMOSC	Sets the cursor in a window.
	DMSWMOSL	Sets window scroll location indicator.
	DMSWMOSR	Sets window reserved lines.
	DMSWMOSW	Sets a window to fixed or variable size.
	DMSWMOSZ	Changes the size of a window.
DMSWMX	DMSWMX	XEDIT/windowing interface.
DMSWQI		Fullscreen CP message handler.
	DMSWQICS	IUCV CONNECT or SEVER the path to the CP *MSGALL Message System Service.
	DMSWQIEX	External internal interrupt handler for the *MSGALL IUCV path.

Module Name	Entry Points	Function
DMSWQM		Queues manager for fullscreen CMS.
	DMSWQMCR	Creates a new queue.
	DMSWQMDL	Deletes a previously defined queue.
	DMSWQMMD	Modifies attributes of a previously defined queue.
	DMSWQMQY	Queues manager query function.
	DMSWQMPR	Purges all the messages currently enqueued on the specified queue.
	DMSWQMPT	Enqueues a message, LIFO or FIFO, on the specified queue.
	DMSWQMGT	Dequeues a message from the specified queue.
DMSWRD	DMSWRD	Processes the WAITREAD command.
DMSWRT	DMSWRT	Processes the WRITE command.
DMSWST	DMSWST	Windowing storage pool allocation.
DMSWTE	DMSWTE	Processes the WAITECB function.
DMSWVC	DMSWVCCL	Clears a virtual screen. All buffers of virtual screen are initialized with defaults.
	DMSWVCCU	Set the cursor at line and column specified in data area or reserved areas.
	DMSWVCRY	Query the virtual screen information: data, cursor, modified fields, virtual screen information, key, and logging information.
DMSWVD	DMSWVDFN	Defines a virtual screen.
	DMSWVDDT	Deletes a virtual screen.
	DMSWVDLA	Deletes all virtual screens.
DMSWVI	DMSWVI	Interface between an application and virtual screen functions.
DMSWVL	DMSWVLPT	Puts content of a virtual screen in a file.
	DMSWVLGT	Gets content of a file into a virtual screen.
	DMSWVLST	Sets logging of a virtual screen on or off.
	DMSWVLPS	Puts contents of a physical screen in a file.
DMSWVQ	DMSWVQSC	Queries a virtual screen control block.
	DMSWVQFD	Queries a field in the virtual screen.
DMSWVS	DMSWVS	Processes the setting of information in a virtual screen.
DMSWVT	DMSWVTTT	Force to update a virtual screen, moving data from queue to a virtual screen.
	DMSWVTIT	Waits on a virtual screen.
	DMSWVTIX	External entry point to wait on a virtual screen.
DMSWVW	DMSWVWS	Writes in the data area of a virtual screen. Called by DMSWVTTT and DMSWVI.
	DMSWVWRS	Writes in the reserved areas of a virtual screen. Called by DMSWVTTT and DMSWVI
DMSWVX	DMSWVX	Processes the moving of data to the virtual screen buffers.
DMSXBG	DMSXBG	Allocates and initializes storage for the XEDIT work area. Processes the XEDIT command.

Module Name	Entry Points	Function
DMSXCG	CDELETE CHANGE COMPRESS COPY COUNT COVERLAY DELETE DUPLICAT EXPAND LOWERCAS MERGE MOVE OVERLAY UPPERCAS RECOVER SHIFT	Processes the subcommands (entry points) listed.
	DMSXCGLS	Locates a string in current line when ETMODE is ON.
DMSXCM	STACK CMS CP	Processes the subcommands (entry points) listed.
DMSXCN	DMSXCN	Arranges compound characters into canonical form; disregards tab characters as special characters.
DMSXCP	DMSXCP	Simulates the VSE EXCP function (VSE SVC 0) in the CMS/DOS environment. EXCP (Execute Channel Program) requests initiation of an I/O operation to a specific logical unit.
DMSXCT	CMSG	Processes the CMSG subcommand.
	CURSOR	Processes the CURSOR subcommand.
	DMSXCTPN	Processes the SET POINT subcommand.
	DMSXCTTE	Processes the SET TERMINAL subcommand.
	DMSXCTSC	Processes the SET SCREEN subcommand.
	EMSG	Processes the EMSG subcommand.
	FILE	Processes the FILE/PFILE subcommand.
	LPREFIX	Processes the LPREFIX subcommand.
	MSG	Processes the MSG subcommand.
	PRESERVE	Processes the PRESERVE subcommand.
	PURGE	Processes the PURGE subcommand.
	READ	Processes the READ subcommand.
	REFRESH	Redisplays the screen.
	RENUM	Processes the RENUM subcommand.
	REPEAT	Processes the REPEAT subcommand.
	RESET	Processes the RESET subcommand.
	RESTORE	Processes the RESTORE subcommand.
	SAVE	Processes the SAVE/PSAVE subcommand.
	TYPE	Processes the TYPE subcommand.
DMSXDC	DMSXDCOD	Scans input from the terminal for a subcommand or macro; operands are decoded and placed in buffers. Executes the MACRO and COMMAND subcommands. Performs synonym substitution.
	DMSXDCSY	Processes the SET SYNONYM subcommand.
	DMSXDCST	Executes multiple synonyms.
	DMSXDCSV	Validates hex strings.
	DMSXDCAT	Adjusts a truncated DBCS string in an operand.

Module Name	Entry Points	Function
DMSXDS	DMSXDSRD	Reads a data set (SAM) from an OS formatted disk.
DMSXED	XEDIT	Processes the XEDIT subcommand; brings a file into the ring of files in storage.
	DMSXEDRT	Removes an edited file from storage (QUIT).
DMSXER	DMSXERMG	Displays an error message in the standard CMS format: DMSxxxnnnc message text....
DMSXFC	DMSXFCNX	Moves the line pointer to the next line.
	DMSXFCUP	Moves the line pointer to the previous line.
	DMSXFCPL	Moves the line pointer to line number "n".
	DMSXFCML	Moves the current line UP1 or NEXT1.
	DMSXFCLA	Locates a line by its address.
	DMSXFCCG	Moves the column pointer to the left.
	DMSXFCLM	Locates a specified string (FIND) in the current line.
	DMSXFCDP	Inserts the column pointer as "_" in a buffer.
	DMSXFCIN	Inserts a new line in the file.
	DMSXFCRL	Replaces a line in the file with a new one.
	DMSXFCSU	Deletes one line from the file.
	DMSXFCRC	Performs string substitution in the current line.
	DMSXFCRM	Performs an overlay function on the current line.
	DMSXFCSP	Handles special characters, ex., tab, backspace.
	DMSXFCDC	Displays SET VERIFY or SET TABS columns.
	DMSXFCLR	Defines the logical record length.
	DMSXFCTR	Defines the truncation column.
	DMSXFCHT	Defines the top of range.
	DMSXFCBT	Defines the end of range.
	DMSXFCGA	Defines the zone left column.
	DMSXFCDR	Defines the zone right column.
	DMSXFCPC	Sets the column pointer in column "n".
	DMSXFCCC	Sets the cursor to column "c" in the file.
	DMSXFCCL	Sets the cursor to line "1" in the file.
	DMSXFCTB	Sets up the tabulation columns.
	DMSXFCPI	Checks if a line has to be spilled.
	DMSXFCLP	Locates first valid SO/SI pair in a string (ETMODE = ON).
	DMSXFCCT	Determines type of character (SO, SI, one-, or two-byte).
	DMSXFCNT	Translation routine.
	DMSXFCRT	Translation routine (excludes DBCS strings from translation).
	DMSXFCTT	Adjusts the DBCS string when spilled or truncated.
	DMSXFCAR	Make adjustments to string being replaced and to replacement string in order to preserve DBCS strings.
	DMSXFCAD	Make adjustments to string being deleted in order to preserve DBCS strings.
	DMSXFCPT	Set the column pointer in column "n"; contains check for left zone when ETMODE is on.

Module Name	Entry Points	Function
DMSXFD	DMSXFDFI DMSXFDSR DMSXFDTG DMSXFDLE DMSXFDLN DMSXFDSD DMSXFDPS DMSXFDSA DMSXFDSB	Writes the file on disk. Serializes the file in storage. Performs target processing. Locates an extended string (a string with arbitrary characters). Locates a named line. Computes string length between delimiters. Parses a string argument. Skips multiple arbitrary characters. Skips multiple blanks.
DMSXFL	DMSXFLST DMSXFLRD DMSXFLWR DMSXFLPT	Determines if a file is in the XEDIT ring, and if so, returns its characteristics. Transfers one (or more) records from XEDIT storage to the calling program. Transfers one (or more) records to XEDIT storage from the calling program. Moves the current line pointer to the record specified by the calling program.
DMSXGT	GET	Process the GET subcommand.
DMSXIN	DFPKSYN DMSXINTF DMSXINLA DMSXINLD DMSXINLX DMSXINOP	Sets default PF keys and synonyms. Initializes a file descriptor block. Aborts the profile macro if an error occurs during LOAD. Processes the LOAD subcommand. Processes and explicit LOAD from the profile macro. Handles XEDIT command options.
DMSXIO	 DMSXIORD DMSXIOWR DMSXIOMG	Performs I/O at the terminal. Reads at the terminal. Writes at the terminal. Displays message pending in the message line.
DMSXMA	DMSXMAOP DMSXMAEX DMSXMARD DMSXMARS	Executes XEDIT macros (files written in EXEC 2 language or REXX language). Executes subcommand from XEDIT macros. States existence of a macro and reads it. Releases a macro from free storage.
DMSXMB	DMSXMBIN DMSXMBRD DMSXMBWR	Reads the member into storage. Reads the directory of the library and locates a member. Writes a directory and member into a library.
DMSXMC	CFIRST CLAST CLOCATE LEFT RIGHT DMSXMCVR	Processes the CFIRST subcommand. Processes the CLAST subcommand. Processes the CLOCATE subcommand. Processes the LEFT subcommand. Processes the RIGHT subcommand. Processes the SET VERIFY subcommand.
DMSXMD	 INPUT ADD REPLACE CREPLACE CINSERT	Processes the subcommands (entry points) listed.

Module Name	**Entry Points**	**Function**
DMSXML	BACKWARD BOTTOM DOWN FIND FINDUP FORWARD FUP LOCATE NEXT NFIND NFINDUP NFUP TOP UP	Processes the subcommands (entry points) listed.
DMSXMS	DMSXMS	Arranges records within a file in a descending or ascending sequential order (SORT macro).
DMSXPO	POWERINP	Allows easy input mode for script users.
DMSXPT	PUT	Processes the PUT subcommand.
	PUTD	Processes the PUTD subcommand.
	DMSXPTER	Erases the temporary file used by GET/PUT(D).
DMSXPX	DMSXPXCL	Calls prefix macros that need to be cleared.
	DMSXPXDC	Decodes prefix subcommands and macros.
	DMSXPXEX	Executes prefix subcommands and macros.
	DMSXPXPN	Sets a prefix in the pending list.
	DMSXPXRS	Resets an entry in the pending list.
DMSXQR	QUERY	Contains the QUERY/TRANSFER subcommands.
	TRANSFER	Stacks variable from the editor.
	DMSXQRCT	Handles QUERY CTLCHAR (control character).
	DMSXQRPT	Handles QUERY POINT.
	DMSXQRCL	Handles QUERY COLOR.
	DMSXQRPY	Handles QUERY PREFIX SYNONYM.
	DMSXQRPK	Handles QUERY PF/PA key.
	DMSXQRPF	Displays PF/PA/Enter Key definition.
	DMSXQRPN	Handles QUERY PENDING.
	DMSXQRRG	Handles QUERY RING.
	DMSXQRSC	Handles QUERY SCREEN.
	DMSXQRTK	Gets the next token in the operand.
DMSXRE	DMSXRE	Processes the RENUM subcommand.
DMSXSC	DMSXSCDP	Dispatches a logical screen.
	DMSXSCFL	Computes to which logical screen belongs a field on the physical screen.
	DMSXSCIM	Builds and displays all the logical screens.
	DMSXSCMB	Manages I/O buffer.
	DMSXSCMF	Manages I/O buffer on fullword boundaries.
	DMSXSCPR	Checks if the cursor is in a protected area.
	DMSXSCCP	Prints an image of the virtual screen (COPYKEY function).
	DMSXSCRV	Displays a line in the command line.
	DMSXSCCS	Sets the cursor on the screen.
	DMSXSCNK	Processes the NULLKEY.

Module Name	Entry Points	Function
DMSXSD	DMSXSDCS	Builds VSBD's for CTLCHARs.
	DMSXSDLS	Builds a logical screen block.
	DMSXSDSC	Builds a logical screen.
	DMSXSDPH	Builds the virtual screen descriptive blocks for fields of screen.
	DMSXSDLN	Builds a line to be displayed.
	DMSXSDML	Moves a line from screen buffer in the file.
	DMSXSDMS	Builds the scale line.
	DMSXSDSR	Scans a line for CTLCHAR.
	DMSXDSLA	Adjusts a line for temporary SO and SI (and attributes).
DMSXSE	DMSXSERA	Handles the SET RANGE subcommand.
	SET	Handles the SET subcommand.
DMSXSF		
	DMSXSFRS	Processes the SET RESERVED subcommand.
	DMSXSFCT	Processes the SET CTLCHAR subcommand.
	DMSXSFMG	Processes the SET MSGLINE subcommand.
	DMSXSFCR	Processes the SET COLOR subcommand.
	DMSXSFSL	Processes the SET SELECT subcommand.
	DMSXSFDM	Decodes a line number (M ± n).
	DMSXSFNT	Gets the next token.
DMSXSS	SOS	Processes the SOS subcommand.
	DMSXSSTB	Tabs backward in the file on the command line.
	DMSXSSTF	Tabs forward in the file on the command line.
DMSXST	DMSXSTLG	Gets a free line in storage.
	DMSXSTNB	Computes the number of free lines available.
	DMSXSTCP	Combines free lines in one free block.
	DMSXSTEX	Dynamically extends the storage for the files.

Module Name	Entry Points	Function
DMSXSU	DMSXSUQT	Obtains CP terminal BRKKEY setting.
	DMSXSUVR	Editing supervisor.
	DMSXSUPE	Executes the profile macro.
	QUIT	Executes the QUIT subcommand.
	DMSXSUFL	Flushes subcommand execution if no more save area.
	DMSXSUNP	No operation (used when a macro ends).
	DMSXSU	Redisplays the last input in the input area.
	DMSXSUIG	Maintains file integrity on multiple windows.
	DMSXSUTY	Types the current line.
	DMSXSUEF	Types "EOF".
	DMSXSUTF	Types "TOF".
	DMSXSUTE	Types "TOF" or "EOF".
	DMSXSUTP	Checks displacement to a target line.
	DMSXSUNC	Types "NO CHANGE".
	DMSXSUNF	Types "NOT FOUND".
	DMSXSUPR	Checks for prefix subcommand or macro waiting.
	DMSXSULG	Computes line length.
	DMSXSUEX	Executes a subcommand.
	DMSXSUCK	Checks if fname ftype fmode are valid.
	DMSXSUTS	Computes autosave identification.
	DMSXSUCN	Performs EBCDIC-binary conversion.
	DMSXSUCC	Performs binary-EBCDIC conversion.
	DMSXSUCH	Performs EBCDIC-hexadecimal conversion.
	DMSXSUHC	Performs hexadecimal-EBCDIC conversion.
	DMSXSULK	Checks coherency between file and logical screen.
	DMSXSURV	Redisplays the last entry in the input area.
	DMSXSUPK	Executes PFKEY/PA2/PA3/Enter Key.
	DMSXSUQM	Executes ? command.
	PQUIT	Removes one file from the ring of files in storage (protected QUIT).
DMSXTB	DMSXTBHC	Address of hash code table.
	DMSXTBRQ	Address of subcommand table.
DMSXTE		Contains the second half of the EXTRACT subcommand.
	DMSXTECU	Assigns CURLINE settings to EXEC 2/REXX variables.
	DMSXTEEX	Performs an EXECCOMM.
	DMSXTERG	Assigns ring settings to EXEC 2/REXX variables.
	DMSXTEEN	Assigns enter settings to EXEC 2/REXX variables.
	DMSXTEFD	Finds next delimiter.
	DMSXTEPA	Parses an argument.
	DMSXTEVR	Assigns VERIFY settings to EXEC 2/REXX variables.
	DMSXTECL	Assigns COLOR settings to EXEC 2/REXX variables.
	DMSXTESC	Assigns SCREEN settings to EXEC 2/REXX variables.
	DMSXTEPS	Assigns PREFIX SYNONYM settings to EXEC 2/REXX variables.
	DMSXTERS	Assigns RESERVED settings to EXEC 2/REXX variables.
	DMSXTEPK	Assigns PF/PA Key settings to EXEC 2/REXX variables.
	DMSXTEHK	Handles setting PF/PA Key values for a specified key.
	DMSXTEPT	Assigns POINT settings to EXEC 2/REXX variables.
	DMSXTESY	Assigns SYNONYM settings to EXEC 2/REXX variables.
	DMSXTEPD	Assigns PENDING settings to EXEC 2/REXX variables.
	DMSXTECC	Assigns CTLCHAR settings to EXEC 2/REXX variables.
	DMSXTEFP	Parses input up to the end of input, or delimiter, or a blank.
	DMSXTEGS	Gets storage from buffer DMSFREED in DMSXTR.
	DMSXTESH	Sets up SHVBLOCK.
	DMSXTETB	Assigns TABS settings to EXEC 2/REXX variables.
	DMSXTEBK	Assigns BRKKEY settings to EXEC 2/REXX variables.

Module Name	Entry Points	Function
DMSXTF	DMSXTF	Filetype descriptor table.
DMSXTR		Contains the EXTRACT subcommand.
	EXTRACT	Assigns editor settings to EXEC 2/REXX variables.
	DMSXTRPA	Performs parse and uppercase option.
	DMSXTRSE	Sets up SHVBLOCK for variable not requiring a special routine to compute variable value.
	DMSXTRTS	Performs table search for option name.
DMSXUP	DMSXUPCK	Checks for proper serialization.
	DMSXUPAT	Applies one update file to the source file.
	DMSXUPCT	Handles CNTRL and AUX files for multi-level update.
	DMSXUPBL	Builds the update file (subcommands SAVE or FILE).
	DMSXUPDL	Handles deleted lines in the source file.
	DMSXUPR2	Builds error messages.
DMSXWS	DMSXWSWD	Handles vscreen and window setup and allocates image buffer.
DMSZAP	DMSZAP	Processes the ZAP command. Provides a facility to maintain CMS LOADLIB members as written by the CMS command LKED.
DMSZAT	DMSZAT	Defines 8K-bytes of transient area.
DMSZIN	DMSZIN	Defines the end of the CMS initialization modules in user storage.
DMSZIT	DMSZIT	Defines the end of the CMS nucleus in user storage.
DMSZNR	DMSZNR	Defines the end of NUCON (DMSNUC).
DMSZUS	DMSZUS	Defines the start of the user area.

Part 4: CMS Diagnostic Aids

This part contains the following information:

- A list of devices supported by a CMS virtual machine
- DMSFREx Error Codes
- Abend Codes.

Chapter 16. Supported Devices

Figure 32 indicates those devices that are supported by a CMS machine.

Virtual IBM Device Type	Virtual Address[6]	Symbolic Name (default)	Device Use
3210, 3215, 1052, 3066, 3270	cuu[7]	CON1	System console
2314, 2319, 3310, 3330, 3340, 3350, 3370, 3375, 3380	190	DSK8	CMS System disk (read-only)
	191[8]	DSK1	Primary disk (user files)
	cuu	DSK2	Minidisk (user files)
	cuu	DSK3	Minidisk (user files)
	192	DSK4	Minidisk (user files)
	cuu	DSK5	Minidisk (user files)
	cuu	DSK6	Minidisk (user files)
	cuu	DSK7	Minidisk (user files)
	19E	DSK9	Minidisk (user files)
	cuu	DSK0	Minidisk (user files)
	cuu	DSKH	Minidisk (user files)
	cuu	DSKI	Minidisk (user files)
	cuu	DSKJ	Minidisk (user files)
	cuu	DSKK	Minidisk (user files)
	cuu	DSKL	Minidisk (user files)
	cuu	DSKM	Minidisk (user files)
	cuu	DSKN	Minidisk (user files)
	cuu	DSKO	Minidisk (user files)
	cuu	DSKP	Minidisk (user files)
	cuu	DSKQ	Minidisk (user files)
	cuu	DSKR	Minidisk (user files)
	cuu	DSKT	Minidisk (user files)
	cuu	DSKU	Minidisk (user files)
	cuu	DSKV	Minidisk (user files)
	cuu	DSKW	Minidisk (user files)
	cuu	DSKX	Minidisk (user files)
2540, 2501, 3505	00C	RDR1	Virtual reader
2540, 3525	00D	PCH1	Virtual punch
1403, 1443, 3203, 3211, 3262, 3800, 3289-4, 4245, 4248	00E	PRN1	Line printer

Figure 32 (Part 1 of 2). Devices Supported by a CMS Virtual Machine

Virtual IBM Device Type	Virtual Address[6]	Symbolic Name (default)	Device Use
2401, 2402, 2403, 2415, 2420, 3410, 3411, 3420, 3430, 3480, 8809, 3422	180-187 288-28F	TAP0-TAP7 TAP8-TAPF	Tape drives

Figure 32 (Part 2 of 2). Devices Supported by a CMS Virtual Machine

[6] The device addresses shown are those that are preassembled into the CMS resident device table. These need only be modified and a new device table made resident to change the addresses.

[7] The virtual address of the system console may be any valid multiplexer address.

[8] 191 is the default user-accessed A-disk unless it is dynamically changed by an ACCESS at CMS initial program load (IPL).

Chapter 17. DMSFREX Error Codes

Error Codes from DMSFREE, DMSFRES, and DMSFRET

A nonzero return code upon return from DMSFREE, DMSFRES, or DMSFRET indicates that the request could not be satisfied. Register 15 contains this return code, indicating which error occurred. The following codes apply to the DMSFREE, DMSFRES, and DMSFRET macros.

Code Error

1 (DMSFREE) Insufficient storage space is available to satisfy the request for free storage. In the case of a variable request, even the minimum request could not be satisfied.

2 (DMSFREE or DMSFRET) User storage pointers destroyed.

3 (DMSFREE, DMSFRET, or DMSFRES) Nucleus storage pointers destroyed.

4 (DMSFREE) An invalid size was requested. This error exit is taken if the requested size is not greater than zero. In the case of variable requests, this error exit is taken if the minimum request is greater than the maximum request. (However, the latter error is not detected if DMSFREE is able to satisfy the maximum request.)

5 (DMSFRET) An invalid size was passed to the DMSFRET macro. This error exit is taken if the specified length is not positive.

6 (DMSFRET) The block of storage that is being released was never allocated by DMSFREE. Such an error is detected if one of the following errors is found:

- The block does not lie entirely inside either the free storage area in low-storage or the user program area between FREELOWE and FREEUPPR.

- The block crosses a page boundary that separates a page allocated for USER storage from a page allocated for NUCLEUS type storage.

- The block overlaps another block already on the free storage chain.

7 (DMSFRET) The address given for the block being released is not on a doubleword boundary.

8 (DMSFRES) An invalid request code was passed to the DMSFRES routine. Since all request codes are generated by the DMSFRES macro, this error code (8) should never appear.

> 8 An unexpected and unexplained error has occurred in the free storage management routine.

Chapter 18. Abend Processing

When CMS abnormally terminates, the following steps are taken:

1. After checking for any SPIE, STXIT PC, STAE, or STXIT AB exits that apply, CMS calls DMSABN, the abend recovery routine.

2. Before typing out any abend message at the terminal, DMSABN, the abend recovery routine, checks for any abend exit routines, set by the ABNEXIT macro.

3. If a list of exit routines exists, the current abend exit routine (that is, the last one set) gains control. If no abend exit routines exist, CMS abend recovery occurs.

Abend Exit Routine Processing

An abend exit routine may be established to intercept abends before CMS abend recovery begins. You must provide the proper entry and exit linkage for this abend exit routine. See the ABNEXIT macro in the *VM/SP CMS Macros and Functions Reference* for details on the register contents when the routine receives control.

The abend exit routine receives control with the nucleus protect key and is disabled for interrupts. Information about the abend is available to the exit routine in the DMSABW CSECT in DMSNUC. The address of this area is passed to the exit routine via register 1. In addition to the information currently available in DMSABW, a fullword specified on the ABNEXIT macro contains information for the exit's own purposes. ABUWRD is the name of the fullword containing the information the user enters in the UWORD parameter of the ABNEXIT macro.

An abend exit routine may choose to avoid CMS abend recovery and continue processing normally. To do this, the exit must issue the ABNEXIT RESET macro. This tells CMS to clear the abend condition. The exit routine may also return to CMS to continue abend processing.

If the exit routine returns to CMS and another abend exit routine exists, it is given control next. Each exit on the list is given control in sequence until all the exits have been given control or until an exit chooses to avoid CMS abend recovery, by issuing ABNEXIT RESET, and continues processing.

If a program check occurs in the exit routine and ABNEXIT RESET was not issued in this exit routine, DMSABN gives control to the next exit routine on the list. If no other exit routine exists, CMS abend recovery occurs.

You cannot set or clear abend exit routines in an abend exit routine. You can reset an abend exit routine only in an exit routine.

CMS Abend Recovery

If no abend exit routine exists or if the abend exit routine returns to CMS to continue abend processing, DMSABN types out the abend message followed by the line:

```
CMS
```

This line indicates to you that the next command can be entered.

Options available to you are:

- Issue the DEBUG command. DMSABN passes control to DMSDBG to make the facilities of DEBUG available. DEBUG's PSW and registers are as they were at the time the recovery routine was invoked. In DEBUG mode, you may alter the PSW or registers. Then, type GO to continue processing, or type RETURN to return to DMSABN. DMSABN continues the abend recovery.

- Issue any command (other than DEBUG). DMSABN performs its abend recovery function and passes control to DMSINT to execute the command that was typed in.

The abend recovery function performs the following functions, in sequence:

1. Clears the console input buffer and program stack.

2. Terminates all VMCF activity.

3. Reinitializes the work area stack for reentrant CMS nucleus modules.

4. Reinitializes the SVC handler, DMSITS, and frees all stacked save areas.

5. Clears the auxiliary directories, if any. Invokes "FINIS * * *", to close all files, and to update the master file directory.

6. Frees storage, if the DMSEXT module is in virtual storage.

7. Zeroes out the MACLIB directory pointers.

8. Frees the CMS work area, if the CMS subset was active.

9. Frets the RLDDATA buffer, used by the CMS loader to retain relocation information for the GENMOD process, if it is still allocated.

10. Issues the STAE, SPIE, TTIMER, and STAX macros to cancel any outstanding OS exit routines. Frees any TXTLIB, MACLIB, or LINK tables.

11. Calls with a purge PLIST, all nucleus extensions that have the "SERVICE" attribute defined.

12. Drops all nucleus extensions that do not have the "SYSTEM" attribute. Also drops any nucleus extensions that are in type user storage.

13. Drops all SUBCOM SCBLOCKS that do not have the "SYSTEM" attribute.

14. Frees console path and device entry control blocks.

15. Drops all storage resident execs that do not have the "SYSTEM" attribute.

16. Clears all immediate commands that are not nucleus extensions with the "SYSTEM" attribute; returns all associated free storage.

17. Calls DMSCLN to zero out the userword of the SRPI command.

18. Calls DMSWITAB to delete all windows and vscreens that do not have the "SYSTEM" attribute.

19. Resets the storage keys for the whole virtual machine, except the nonshared pages, according to FREETAB. Saves the setting for KEYPROTECT.

20. Zeroes out all FCB, DOSCB, and LABSECT pointers.

21. Frees all storage of type user.

22. Restores the setting for KEYPROTECT.

23. Zeroes out all interrupt handler pointers in IOSECT.

24. Turns the SVCTRACE command off.

25. Closes the virtual punch and printer; closes the virtual reader with the HOLD option.

26. Reinitializes the VSE lock table used by CMS/DOS and CMS/VSAM.

27. Zeroes out all OS loader blocks, and frees the FETCH work area.

28. Cleans up the CMS IUCV environment based on the existence of the CMS id block.

29. Clears all ABNEXIT set and returns storage.

30. Computes the amount of system free storage that should be allocated and compares this amount with the amount of free storage actually allocated. Types a message to the user if the two amounts are unequal.

31. Issues a STRINIT and releases any pages remaining in the flush list via a call to DMSPAGFL, if all storage is accounted for.

After abend recovery has completed, control passes to DMSINT at entry point DMSINTAB to process the next command.

Unrecoverable Termination--The Halt Option of DMSERR

There are certain times, such as when the SVC handler's pointers are modified, that the system can neither continue processing nor try to recover. In these cases, DMSERR with the option HALT=YES is specified to cause a message to be typed out, after which a disabled wait state PSW is loaded unless the NUCON field AUSERRST has been loaded.

The valid address contained in AUSERRST is assumed to be the address of an error recovery routine and will be directly branched to. The initialization routines of an application running under CMS must set this address to point to a module that might, for example, request a dump and then issue an IPL command. If the IPL command is

```
IPL CMS PARM AUTOCR
```

and the PROFILE EXEC on virtual disk 191 invokes reinitialization, the application has the capability of automatic recovery. This capability is valuable for CMS service virtual machines that run permanently disconnected and are required to stay operational.

In CP mode, the programmer can examine the PSW, whose address field contains the address of the instruction following the call to the DMSERR macro. The programmer can also examine all the registers, which are as they were when the DMSERR macro was invoked.

Figure 33 lists the CMS ABEND codes and describes the cause of the abend and the action required.

Abend Code	Module Name	Cause of Abend	Action
001	DMSSCT	The problem program encountered an input/output error processing an OS macro. Either the associated DCB did not have a SYNAD routine specified or the I/O error was encountered processing an OS CLOSE macro.	Message DMSSCT120S indicates the possible cause of the error. Examine the error message and take the action indicated.

Figure 33 (Part 1 of 5). CMS Abend Codes

Abend Code	Module Name	Cause of Abend	Action
034	DMSVIP	The problem program encountered an I/O error while processing a VSAM action macro under VSE for which there is no OS equivalent. An internal error occurred in a VSE/VSAM routine.	Refer to *VSE/VSAM Messages and Codes* to determine the cause of the VSAM error.
035	DMSVIP	An error occurred in VSE/VSAM processing while running an OS/VSAM program, for which there is no equivalent OS/VSAM error code.	Refer to the VSE/VSAM documentation for the error and return codes indicated in the CMS error message preceding the ABEND.
09F	DMSITP	A vector operation exception (program interrupt code X'19') occurred at a specified location.	Type DEBUG to examine the PSW and registers at the time of the exception. Use the CP DISPLAY command to examine the vector registers. Refer to the *IBM System/370 Vector Operations*, SA22-7125 for a description of the vector operation exception.
0Cx	DMSITP	The specified hardware exception occurred at a specified location. "x" is the type of exception: **x Type** 1 OPERATION 2 PRIVILEGED OPERATION 3 EXECUTE 4 PROTECTION 5 ADDRESSING 6 SPECIFICATION 7 DATA 8 FIXED-POINT OVERFLOW 9 FIXED-POINT DIVIDE A DECIMAL OVERFLOW B DECIMAL DIVIDE C EXPONENT OVERFLOW D EXPONENT UNDERFLOW E SIGNIFICANCE F FLOATING-POINT DIVIDE	Type DEBUG to examine the PSW and registers at the time of the exception.
0D3	DMSITP	A special operation exception (program interrupt code X'13') occurred at a specified location.	Type DEBUG to examine the PSW and registers at the time of the exception.

Figure 33 (Part 2 of 5). CMS Abend Codes

Abend Code	Module Name	Cause of Abend	Action
0E0	DMSITP	A hardware exception occurred at a specified location.	Type DEBUG to examine the PSW and registers at the time of the exception. Bytes 2 and 3 of the BC Mode Program Old PSW are the program interrupt code. This indicates the type of exception that occurred. Refer to the *IBM System/370 Principles of Operation*, GA22-7000 or the *IBM System/370 Vector Operations*, SA22-7125 for a description of the hardware exception.
0F0	DMSITS	Insufficient free storage is available to allocate a save area for an SVC call.	If the abend was caused by an error in the application program, correct it; if not, use the CP DEFINE command to increase the size of virtual storage and then restart CMS.
0F1	DMSITS	An invalid halfword code is associated with SVC 203.	Enter DEBUG and type GO. Execution continues.
0F2	DMSITS	The CMS nesting level of 20 has been exceeded.	None. Abend recovery takes place when the next command is entered.
0F3	DMSITS	CMS SVC (202 or 203) instruction was executed and provision was made for an error return from the routine processing the SVC.	Enter DEBUG and type GO. Control returns to the point to which a normal return would have been made.
0F4	DMSITS	The DMSKEY key stack overflowed.	Enter DEBUG and type GO. Execution continues and the DMSKEY macro is ignored.
0F5	DMSITS	The DMSKEY key stack overflowed.	Enter DEBUG and type GO. Execution continues and the DMSKEY macro is ignored.
0F6	DMSITS	The DMSKEY key stack was not empty when control returned from a command or function.	Enter DEBUG and type GO. Control returns from the command or function as if the key stack had been empty.
0F7	DMSFRE	Occurs when TYPCALL=SVC (the default) is specified in the DMSFREE or DMSFRET macro.	When a system abend occurs, use DEBUG to attempt recovery.

Figure 33 (Part 3 of 5). CMS Abend Codes

Abend Code	Module Name	Cause of Abend	Action
0F8	DMSFRE	Occurs when TYPCALL=BALR is specified in the DMSFREE or DMSFRET macro devices.	When a system abend occurs, use DEBUG to attempt recovery.
101	DMSSVN	The wait count specified in an OS WAIT macro was larger than the number of ECBs specified.	Examine the program for excessive wait count specification.
104	DMSVIB	The OS interface to VSE/VSAM is unable to continue execution of the problem program.	See the additional error message accompanying the abend message, correct the error, and reexecute the program.
155	DMSSLN	Error during LOADMOD after an OS LINK, LOAD, XCTL, or ATTACH. The compiler switch is on.	See the last LOADMOD (DMSMOD) error message for error description. In the case of an I/O error, recreate the module. If the module is missing, create it.
15A	DMSSLN	Severe error during load (phase not found) after an OS LINK, LOAD, XCTL, or ATTACH. The compiler switch is on.	See last LOAD error message (DMSLIO) for the error description. In the case of an I/O error, recreate the text deck or TXTLIB. If either is missing, create it.
160	DMSXSU	Occurs when XEDIT cannot allocate a save area to a called routine.	None. Abend recovery takes place when the next command is entered.
174	DMSVIB	The OS interface to VSE/VSAM is unable to continue execution of the problem program.	See the additional error message accompanying the abend message, correct the error, and reexecute the program.
177	DMSVIB DMSVIP	The OS interface to VSE/VSAM is unable to continue execution of the problem program.	See the additional error message accompanying the abend message, correct the error, and reexecute the program.
240	DMSSVT	No work area was provided in the parameter list for an OS RDJFCB macro.	Check RDJFCB specification.
400	DMSSVT	An invalid or unsupported form of the OS XDAP macro was issued by the problem program.	Examine program for unsupported XDAP macro or for SVC 0.
500	DMSTLB	A block count error was detected when reading an SL tape. User replied 'cancel' to message 425R or the user's program contained a block count error routine that returned a code of 0 under OS simulation.	Find out what caused the block count error. Then reload CMS and rerun the job.

Figure 33 (Part 4 of 5). CMS Abend Codes

Abend Code	Module Name	Cause of Abend	Action
704	DMSSMN	An OS GETMAIN macro (SVC 4) was issued specifying the LC or LU operand. These operands are not supported by CMS.	Change the program so that it specifies allocation of only one area at a time.
705	DMSSMN	An OS FREEMAIN macro (SVC 5) was issued specifying the L operand. This operand is not supported by CMS.	Change the program so that it specifies the release of only one area at a time.
804 80A	DMSSMN	An OS GETMAIN macro (804 - SVC 4, 80A - SVC 10) was issued that requested either zero bytes of storage or more storage than was available.	Check the program for a valid GETMAIN request. If more storage was requested than was available, increase the size of the virtual machine and retry. If you run out storage while trying to acquire a large GETMAIN area and if the size of your virtual machine is above the start of the CMS nucleus, you should IPL a CMS system generated at a higher virtual address than the one you are using.
905 90A	DMSSMN	An OS FREEMAIN macro (905 - SVC 5, 90A - SVC 10) was issued specifying an area to be released whose address was not on a doubleword boundary.	Check the program for a valid FREEMAIN request; the address may have been incorrectly specified or modified.
A05 A0A	DMSSMN	An OS FREEMAIN macro (A05 - SVC 5, A0A - SVC 10) was issued specifying an area to be released that overlaps an existing free area.	Check the program for a valid FREEMAIN request; the address and/or length may have been incorrectly specified or modified.

Figure 33 (Part 5 of 5). CMS Abend Codes

Appendixes

Appendix A. CMS Macro Library

The following is a list and brief description of the CMS macros supported for use by application programs.

CMS Macro	Function
ABNEXIT	Sets or clears abend exit routines.
ADDENTRY	Tells the SRPI to notify a program when COPROC communications end.
APPLMSG	Accesses and displays messages from an application repository file.
BATLIMIT	Table of CPU, punch, and printer limits for user jobs running under CMS batch.
CMSDEV	Obtains VM/SP device characteristic information and places it in a user-provided buffer.
CMSIUCV	Initializes or terminates IUCV communications with another IUCV program or with CP.
CMSLEVEL	Defines the value of 'release number' of the feature or licensed program returned by QUERY CMSLEVEL. Refer to the CMSLEVEL macro for more information.
COMPSWT	Sets the compiler switch on or off. Refer to the *VM/SP CMS Macros and Functions Reference.*
CONSOLE	Performs CMS fullscreen I/O services.
CPRB	Allocates CPRB storage or generates DSECT.
CQYSECT	Maps console path and/or device information to a user's buffer specified on the CONSOLE OPEN or QUERY function.
CSMRETCD	IBM Cooperative Processing return code equates.
DELENTRY	Drops entry names previously placed on the list via ADDENTRY.
DISPW	Generates the calling sequence for the display terminal interface. Refer to the *VM/SP CMS for System Programming.*
DMSABN	Abend the virtual machine. Refer to the *VM/SP CMS for System Programming.*
DMSEXS	Execute an instruction without nucleus protection. Refer to the *VM/SP CMS for System Programming.*
DMSFREE	Gets free storage. Refer to the *VM/SP CMS for System Programming.*
DMSFRET	Releases free storage. Refer to the *VM/SP CMS for System Programming.*
DMSFST	Sets up a file status table for a given file. Refer to the *VM/SP CMS for System Programming.*

CMS Macro	Function
DMSKEY	Sets nucleus protection on or off. Refer to the *VM/SP CMS for System Programming.*
EPLIST	DSECT to map extended PLIST passed in register 0.
FSCB	Sets up a file system control block. Refer to the *VM/SP CMS Macros and Functions Reference.*
FSCBD	DSECT that describes fields in CMS PLIST for related commands.
FSCLOSE	Closes a file. Refer to the *VM/SP CMS Macros and Functions Reference.*
FSERASE	Erases a file. Refer to the *VM/SP CMS Macros and Functions Reference.*
FSOPEN	Opens a file. Refer to the *VM/SP CMS Macros and Functions Reference.*
FSPOINT	Executes the CMS POINT function.
FSREAD	Reads a record from a file. Refer to the *VM/SP CMS Macros and Functions Reference.*
FSSTATE	Checks for an existing file. Refer to the *VM/SP CMS Macros and Functions Reference.*
FSWRITE	Writes a record into a disk file. Refer to the *VM/SP CMS Macros and Functions Reference.*
HNDEXT	Handles external and timer interrupts. Refer to the *VM/SP CMS Macros and Functions Reference.*
HNDINT	Handles interrupt on devices. Refer to the *VM/SP CMS Macros and Functions Reference.*
HNDIUCV	Initializes or terminates a virtual machine's IUCV communications.
HNDSVC	Handles SVCs. Refer to the *VM/SP CMS Macros and Functions Reference.*
IMMBLOK	Maps the Immediate Command Name Block.
IMMCMD	Declares, clears, and queries Immediate commands.
LANGBLK	Generates a language control block for an application.
LINEDIT	Types a line to the terminal. Refer to the *VM/SP CMS Macros and Functions Reference.*
LINERD	Reads a line of input from the terminal. Refer to *VM/SP CMS Macros and Functions Reference.*
LINEWRT	Reads a line of input from the terminal. Refer to *VM/SP CMS Macros and Functions Reference.*
NUCON	Generates a DSECT CMS nucleus constant area.
PARSECMD	Parses command arguments. Refer to the *VM/SP CMS Macros and Functions Reference.*
PARSERCB	Generates a parser control block DSECT. Refer to the *VM/SP CMS Macros and Functions Reference.*
PARSERUF	Generates a mapping for the user token validation function parameter control block. Refer to the *VM/SP CMS Macros and Functions Reference.*
PRINTL	Prints a line on the printer. Refer to the *VM/SP CMS Macros and Functions Reference.*

CMS Macro	Function
PUNCHC	Punches a card. Refer to the *VM/SP CMS Macros and Functions Reference.*
PVCENTRY	Generates a DSECT mapping for the parser validation code table. Refer to the *VM/SP CMS Macros and Functions Reference.*
RDCARD	Reads a card from the reader. Refer to the *VM/SP CMS Macros and Functions Reference.*
RDTAPE	Reads a record from tape. Refer to the *VM/SP CMS Macros and Functions Reference.*
RDTERM	Reads a record from the terminal. Refer to the *VM/SP CMS Macros and Functions Reference.*
REGEQU	Generates symbolic register equates. Refer to the *VM/SP CMS Macros and Functions Reference.*
SCBLOCK	Maps the subcommand block.
SENDREQ	Sends service requests to servers.
SHVBLOCK	Maps the shared variable block.
STRINIT	Initializes storage. Refer to the *VM/SP CMS Macros and Functions Reference.*
TAPECTL	Positions a tape. Refer to the *VM/SP CMS Macros and Functions Reference.*
TAPESL	Processes standard HDR1 and EOF1 tape labels.
TEOVEXIT	Sets up and clears a CMS tape end-of-volume exit.
TRANTBL	Generates a DSECT mapping of system translation tables. Refer to the *VM/SP CMS for System Programming.*
TVSPARMS	Sets tape volume switching parameters for DMSTVS. Refer to the *VM/SP CMS for System Programming.*
USERSECT	Maps the user work area.
WAITD	Waits until the next interrupt occurs for the specified device. Refer to the *VM/SP CMS Macros and Functions Reference.*
WAITECB	Waits on an ECB or a list of ECBs.
WAITT	Waits until all pending I/O to the terminal has completed. Refer to the *VM/SP CMS Macros and Functions Reference.*
WRTAPE	Writes a record to tape. Refer to the *VM/SP CMS Macros and Functions Reference.*
WRTERM	Writes a record to the terminal. Refer to the *VM/SP CMS Macros and Functions Reference.*

Appendix B. CMS/DOS Macro Library

CMS contains a DOS macro library with the following significant entries. A more complete list may be obtained by looking at the DOSMACRO EXEC; this EXEC produces a list of all the macros in the DOS library.

Macro	Function
CCB	Generates the DOS/VS command control block.
COMRG	Returns address of background partitions communication region; expands to SVC 33.
EOJ	Normal processing termination; expands to SVC 0.
OPENR	Activates a data file; simulated by DMSOR1, DMSOR2, DMSOR3.
STXIT	Provides/terminates supervisor linkage to user's program check routines; simulated by DMSDOS.
IKQACB	DSECT for VSAM ACB (access method control block).
IKQEXLST	DSECT for VSAM EXLST control block (contains addresses of user exit routines).
IKQRPL	DSECT for VSAM RPL (request parameter list control block).
ABTAB	DSECT of abnormal termination option table.
FICL	DSECT, CMS/DOS first in class table.
NICL	DSECT, CMS/DOS number in class table.
PUBOWNER	DSECT, physical unit block ownership table.
ANCHTAB	DSECT, DOS/VS anchor table.
FCHTAB	DOS/VS fetch table containing fetch/load parameter list.
MAPPUB	DSECT defines fields of CMS/DOS physical unit block (PUB).
PUBTAB	DSECT same usage as MAPPUB.
EXCPW	DSECT, work area for DMSXCP routine.
LUBTAB	DSECT for CMS/DOS logical unit block.

Appendix C. CMS/DOS Support Modules

The modules listed below (by phase) make up the CMSBAM segment. The phases and modules (except DMSLBR) retain their VSE identifiers.

Phase	Modules
$IJBLKMD	IJBLKMD
$IJBLBSL	IJBLBSL
$IJGXCP	IJGXCP
$IJGXDI	IJGXDI
$IJGXSDF	IJGXSDF
$IJGXSDU	IJGXSDU
$IJGXSDV	IJGXSDV
$IJGXSDW	IJGXSDW
$IJBLKMD	IJBLKMD
$IJGXSFI	IJGXSFI
$IJGXSRI	IJGXSRI
$IJGXSSR	IJGXSSR
$IJGXSVI	IJGXSVI
$IJJGTOP	IJJGDACX IJJGDAI1 IJJGDAI2 IJJGDAMO IJJGDAMS IJJGDAMX IJJGDAO1 IJJGDAO2 IJJGDAO3 IJJGDAO4 IJJGDAO5 IJJGDARL IJJGDART IJJGDAVC IJJGMFBA IJJGMIOI IJJGMLLM IJJGMMBF IJJGMSOO IJJGMS1O IJJGMTOP IJJGSDBH IJJGSDBS IJJGSDCD IJJGSDCI IJJGSDCI IJJGSDCW IJJGSDFP IJJGSDGC IJJGSDI1 IJJGSDI2 IJJGSDI3 IJJGSDI4 IJJGSDI5 IJJGSDLP IJJGSDMC IJJGSDMF IJJGSDMN IJJGSDMO IJJGSDNV IJJGSDO1 IJJGSDO2 IJJGSDO4 IJJGSDO5 IJJGSDO6 IJJGSDO7 IJJGSDRL IJJGSDSF IJJGSDUL IJJGSDVH IJJGSDW1 IJJGSDW2 IJJGSDW3 IJJGSDW4 IJJGSDXT IJJGVD00 IJJGVD10 IJJGVM00 IJJGVM10
$IJJHCVH	IJJHCCV0 IJJHCVH0 IJJHOPN0 IJJHRDS0 IJJHSRN0 IJJHWDS0
DMSLBR	DMSLBR

Summary of Changes

Summary of Changes for the VM/SP System Logic and Problem Determination Guide Volume 2 (CMS)

Summary of Changes
for LY20-0893-4
for VM/SP Release 5

VM/SP Enhanced Usability

VM/SP now has the following usability enhancements:

- Window functions
- Fullscreen environment for CMS
- A CONSOLE macro.

When CMS is in fullscreen mode, the user can enter commands from anywhere on the physical screen, scroll through data, and log data into files. The CONSOLE macro performs 3270 I/O operations.

Modified CMS Commands

GLOBAL Command

The enhanced GLOBAL command now allows you to list up to 63 (formerly 8) libraries from one of the supported library types (MACLIB, TXTLIB, DOSLIB, or LOADLIB) to be searched for macros, copy files, subroutines, VSE executable phrases, or OS load modules when processing subsequent CMS commands.

LOAD Command

The new HIST option of the LOAD command lets you add comments from TEXT files into MODULE files.

INCLUDE Command

The new HIST option of the INCLUDE command lets you add comments from TEXT files into MODULE files.

Enhancements for EXECs in Storage

VM/SP now has an optional Installation Discontiguous Shared Segment (DCSS) to contain frequently used EXECs and Editor Macros that your installation provides. All users can access the DCSS and share the same executing copy of the EXECs.

Enhanced Interactive Facility/System Profile

The system profile is an EXEC that performs some CMS initialization function previously done in a module. By modifying this EXEC, the system programmer will be able to tailor the CMS environment to suit the installation's needs.

Parsing Facility

The CMS parsing facility parses and translates command name arguments. This lets users enter commands in national languages supported by VM/SP. These languages are: American English, KANJI, Uppercase English, French, German.

To use the parsing facility, you must define command syntax in a special language, the Definition Language for Command Syntax (DLCS). The parsing facility parses a specified command by checking whether command arguments are specified according to the DLCS definition for that command.

Defining command syntax in a DLCS file and using the parsing facility has the following advantages:

1. Syntax checking is unnecessary in programs.

2. Users can invoke programs in their own national language by modifying the DLCS file.

IBM Cooperative Processing for VM/SP

This part of CMS supports IBM Cooperative Processing on a VM/SP System. The support includes:

- A means for work station users to connect to VM/SP to run applications that use VM/SP resources on their work stations.

- An application program interface call the Service Request Programming Interface (SRPI) that allows programmers to write applications for work station (IBM Personal Computer) users.

Device Support

Support has been added for the following device:

- 3422 Magnetic Tape Subsystem

Miscellaneous

Most CMS messages and responses are now in mixed case.

Minor technical and editorial changes have been made throughout this publication.

Summary of Changes for LY20-0893-3 for VM/SP Release 4

Programmable Operator/NCCF Interface

This gives an NCCF operator the ability to operate a distributed VM System. Specifically, the programmable operator function is modified to route messages to the NCCF operator. The NCCF operator can then direct commands to and receive responses from the programmable operator function.

System Product Interpreter Enhancements

This item consists of:

- Two-byte code (TBC) support

  The System Product Interpreter is modified to support processing of two-byte KANJI characters from a fullscreen environment or from a subset of the current XEDIT commands.

- MACLIB-member editing support

  A fullscreen interface to select a member from a MACLIB for editing is provided.

- Usability enhancements

  The usability of XEDIT will be enhance through:

  - Mixed case messages
  - Programmed symbol set support
  - Permitting CP BRKKEY to be turned off
  - A prefix macro for repeated input and automatic indenting.

EXECs in Storage

This support allows the user to load into storage the EXECs that are to remain storage resident. The EXECs specified are loaded, prepared for execution, and remain in storage ready for execution until specifically purged by the user.

Tape Support

This item consists of the following:

- VMFPLC2 modifications

  Serviceability is improved by increasing the patch area and resolving a block problem.

- TAPE command modifications

  Serviceability is improved by increasing the patch area.

- Relocatable loader

  A relocatable loader to load modules as nucleus extensions.

- CMS tape volume switching

  Limited multivolume tape support for OS simulation.

- CMS OS simulation multivolume tape support

  Support for IBM standard-labeled tapes.

- Improved OS standard-labeled tape support

  Tapes created under OS simulation are made more compatible with tapes created created by a native OS system.

Device Support

Support has been added for the following devices:

- 3290 Information Panel
- 3370 Direct Access Storage, Models A2 and B2
- 3480 Magnetic Tape Subsystem
- 4248 Printer.

Miscellaneous

Minor technical and editorial changes have been made throughout this publication.

Summary of Changes for LY20-0893-2 for VM/SP Release 3

Modules DMSQRY and DMSDOS Split

The modules DMSQRY and DMSDOS have been split. DMSQRY was split into the following modules: DMSQRS, DMSQRT, DMSQRU, DMSQRV, DMSQRW, DMSQRX, DMSQRY, and DMSQRZ.

DMSDOS handles CMS/DOS SVC requests. DMSDOS passes control to the appropriate module to handle the SVC. The following modules handle the SVC functions: DMSETR, DMSGMF, DMSGTM, DMSGVE, DMSLCK, DMSLDF, DMSLIC, DMSMCM, DMSRPG, DMSSTX, DMSSUB, DMSSVL, DMSVIS, and DMSXCP.

CMS Enhancements - IUCV

CMS now supports IUCV communication. The two new macros, HNDIUCV and CMSIUCV, enable programs to invoke IUCV functions. These macros also allow the user to specify exits for any IUCV external interrupts that occur on the path. CMS support allows multiple subsystems/applications to use IUCV functions within a virtual machine.

System Product Interpreter Information

The System Product Interpreter processing is described. All modules associated with the System Product Interpreter are documented.

New CMS Commands

CATCHECK command
: A new CMS command that allows VSAM users to invoke the VSE/VSAM Catalog Check Service Aid to verify a catalog structure.

RESERVE command
: A new CMS command that allocates all available blocks of a 512-, 1K-, 2K-, or 4K-byte block formatted minidisk to a unique CMS file. DMSRSV processes this command.

IMMCMD command
: A new CMS command that establishes and cancels immediate commands. This command should be issued only from EXECs. DMSIMM processes this command.

EXECOS command
: A new CMS command that resets the OS and VSAM environments under CMS without returning to the interactive environment. DMSSTG processes this command.

New CMS Macros

ABNEXIT macro
: A new CMS macro that sets or clears abend exit routines. DMSABX processes this macro.

WAITECB macro
: A new CMS macro that waits on an ECB or a list of ECBs. DMSWTE processes this macro.

IMMCMD macro
: A new CMS macro that declares, clears, and queries immediate commands. DMSIMM processes this macro.

TEOVEXIT macro
: A new CMS macro that sets up and clears a CMS tape end-of-volume exit.

New CMS function

DISKID function
: A new CMS function that obtains information on the physical organization of a minidisk - the virtual address, the blocksize, and the offset of the RESERVEd minidisk.

Modified CMS Commands and Macros

NUCXLOAD command
A nucleus extension with the ENDCMD attribute specified receives control at normal end-of-command processing.

FORMAT command
This command now allows you to specify a 512-byte block minidisk.

RDTERM macro
The RDTERM macro with the TYPE=DIRECT attribute specified indicates that a line is to be read directly from the virtual machine console.

IDUMP

VSE IDUMP macro will be simulated by CMS/DOS using the PDUMP support. The IDUMP macro produces a dump containing information about the failing component. The CMS IDUMP support is invoked whenever a licensed program issues the VSE·IDUMP macro.

CMSSEG was Eliminated

CMSSEG was eliminated and the code was merged into the CMS Nucleus.

PROP Enhancements

To support the new enhancements to PROP, the following modules were added: DMSPOL, DMSPOQ, and DMSPOS.

VM/SP 3800 Model 3 Compatibility Support

The SETPRT command has been modified to allow VM/SP users to access the 3800 Model 3 library character sets (LCSs), graphic character modification modules (GRAPHMODs), and the 10 lines-per-inch vertical space option.

Miscellaneous

Minor technical and editorial changes have been made throughout this publication.

Glossary of Terms and Abbreviations

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

- "2305" refers to IBM 2305 Fixed Head Storage, Models 1 and 2.
- "270x" refers to IBM 2701, 2702, and 2703 Transmission Control Units or the Integrated Communications Adapter (ICA) on the System/370 Model 135.
- "FB-512" refers to those IBM DASD devices implementing the fixed-blocked (512-byte blocks) architecture. Specifically, they are the IBM 3310, and the IBM 3370. Current IBM disk storage devices are referred to as count-key-data DASD when it is important to distinguish between count-key-data DASD and FB-512. Otherwise, they are collectively referred to as DASD and disk.
- "3330" refers to the IBM 3330 Disk Storage, Models 1, 2, or 11; the IBM 3333 Disk Storage and Control, Models 1 or 11; and the 3350 Direct Access Storage operating in 3330/3333 Model 1 or 3330/3333 Model 11 compatibility mode.
- "3340" refers to the IBM 3340 Disk Storage, Models A2, B1, and B2, and the 3344 Direct Access Storage Model B2.
- "3350" refers to the IBM 3350 Direct Access Storage Models A2 and B2 in native mode.
- "3370" refers to the IBM 3370 Direct Access Storage Models A1, A2, B1, and B2.
- "3380" refers to the IBM 3380 Storage Facility. Information on the 3380 Storage Facility is for planning purposes only until the availability of the product.
- "3704," "3705," or "370x" refers to the IBM 3704 and 3705 Communications Controllers.
- The term "3705" refers to the 3705 I and the 3705 II unless otherwise noted.
- "2741" refers to the IBM 2741 and the 3767, unless otherwise specified.
- "3270" refers to a series of display devices, namely the IBM 3275, 3276, 3277, 3278, 3279 Display Stations, and the 3290 Information Panel. A specific device type is used only when a distinction is required between device types.
- The term, System/370 processors, is also applicable to 4300 processors and 303x series processors unless indicated otherwise.
- Information about display terminal usage also applies to the IBM 3036, 3138, 3148, and 3158 Display Consoles when used in display mode, unless otherwise noted.
- Any information pertaining to the IBM 3284 or 3286 also pertains to the IBM 3287, 3288 and the 3289 printers, unless otherwise noted.
- "3262" refers to the IBM 3262 Printer, Models 1 and 11. Information on the IBM 3262 Printer, Models 1 and 11, is for Planning purposes only, until the availability of the product.
- "3800" refers to the IBM 3800 Printing Subsystems, Models 1 and 3. A specific device type is used only when a distinction is required between device types.

Unless otherwise noted, the term "VSE" refers to the combination of the DOS/VSE system control program and the VSE/Advanced Functions program product.

In certain cases, the term DOS is still used as a generic term. For example, disk packs initialized for use with VSE or any predecessor DOS or DOS/VS system may be referred to as DOS disks.

The DOS like simulation environment provided under the CMS component of the VM/System Product, continues to be referred to as CMS/DOS.

Bibliography

Here is a list of IBM books that can help you use your system. If you don't see the book you want in this list, you might want to check the *IBM System/370, 30xx, and 4300 Processors Bibliography.*

Prerequisite Publication

Virtual Machine/System Product

Introduction, GC19-6200

Terminal Reference, GC19-6206

System Messages Cross-Reference, SC24-5264

CMS Command Reference, SC19-6209

CMS Macros and Functions Reference, SC24-5284

CMS User's Guide, SC19-6210

Corequisite Publication

Virtual Machine/System Product

Operator's Guide, SC19-6202

CP Command Reference, SC19-6211

CP for System Programming, SC24-5285

CMS for System Programming, SC24-5286

Transparent Services Access Facility Reference, SC24-5287

System Messages and Codes, SC19-6204

OLTSEP and Error Recording Guide, SC19-6205

Interactive Problem Control System Guide, SC24-5260

Service Routines Program Logic, LY20-0890

Data Areas and Control Block Logic Volume 1 (CP), LY24-5220

Data Areas and Control Block Logic Volume 2 (CMS), LY24-5221

Virtual Machine

System Facilities for Programming, SC24-5288

Diagnosis Guide, LY24-5241

Running Guest Operating Systems, SC19-6212

In addition, for EREP processing the following OS/VS Library publications are required:

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

IBM OS/VS, DOS/VSE, VM/370 Environmental Recording Editing and Printing (EREP) Program Logic, SY28-0773.

Supplementary Publications

IBM System/360 Principles of Operation, GA22-6821

IBM System/370 Principles of Operation, GA22-7000

IBM OS/VS, DOS/VS, and VM/370 Assembler Language, GC33-4010

IBM OS/VS and VM/370 Assembler Programmer's Guide, GC33-4021

Miscellaneous Information

CMS/DOS is part of the CMS system and is not a separate system. The term CMS/DOS is used in this manual as a concise way of stating that the DOS simulation mode of CMS is currently active; that is, the CMS command

```
SET DOS ON
```

has been previously issued.

The phrase “CMS file system” refers to disk files that are in CMS's 512-, 800-, 1024-, 2048-, and 4096-byte block format. CMS's VSAM data sets are not included.

The VM/SP Library (Part 1 of 3)

Evaluation

- General Information GC20-1838
- Introduction GC19-6200

Index

- Library Guide, Glossary, and Master Index GC19-6207

Planning

- Planning Guide and Reference SC19-6201
- Running Guest Operating Systems GC19-6212
- Release 5 Guide SC24-5290
- Distributed Data Processing Guide SC24-5241

Installation

- Installation Guide SC24-5237

Applications

- Application Development Guide SC24-5247
- Programmer's Guide to the SRPI for VM/SP SC24-5291

Operation

- Operator's Guide SC19-6202

Reference Summaries

To order all of the Reference Summaries, use order number SBOF-3242

- Commands (General User) SX20-4401
- Commands (Other than General User) SX20-4402
- SP Editor Command Reference Summary SX24-5122
- EXEC 2 Reference Summary SX24-5124
- Sys.Prod Interpreter Reference Summary SX24-5126
- CMS Primer Summary of Commands SX24-5151
- CMS Primer Line-Oriented Summary of Commands SX24-5159
- Problem Reporting Summary (Poster) SX24-5171
- Summary of End Use Tasks and Commands (Poster) SX24-5173

The VM/SP Library (Part 2 of 3)

End Use

- Terminal Reference GC19-6206
- CMS Primer SC24-5236
- CMS Primer for Line-Oriented Terminals SC24-5242
- CMS User's Guide SC19-6210
- CMS Command Reference SC19-6209
- CMS Macros and Functions Reference SC24-5284
- System Product Editor User's Guide SC24-5220
- System Product Editor Command and Macro Reference SC24-5221
- System Product Interpreter User's Guide SC24-5238
- System Product Interpreter Reference SC24-5239
- EXEC 2 Reference SC24-5219
- CP Command Reference SC19-6211
- Quick Reference SX20-4400

Diagnosis

- System Messages and Codes SC19-6204
- System Messages Cross-Reference SC24-5264
- Service Routines Program Logic LY20-0890
- Problem Reporting Guide SC24-5282
- VM Diagnosis Guide LY24-5241
- GCS Diagnosis Reference LY24-5239
- Problem Determination Vol. 1 (CP) LY20-0892
- Data Areas and Control Blocks Vol. 1 (CP) LY24-5220
- Problem Determination Vol. 2 (CMS) LY20-0893
- Data Areas and Control Blocks Vol. 2 (CMS) LY24-5221
- OLTSEP and Error Recording Guide SC19-6205
- VM Problem Determination Reference Information LX23-0347
- VM CP Internal Trace Table (Poster) LX24-5202

The VM/SP Library (Part 3 of 3)

Administration

- VM System Facilities for Programming SC24-5288
- CP for System Programming SC24-5285
- CMS for System Programming SC24-5286
- TSAF Reference SC24-5287
- GCS Command and Macro Reference SC24-5250

Auxiliary Communication Support

- VTAM Installation and Resource Definition SC23-0111
- VTAM Customization SC23-0112
- VTAM Operation SC23-0113
- VTAM Messages and Codes SC23-0114
- VTAM Reference Summary SC23-0135
- VTAM Programming SC23-0115
- VTAM Diagnosis Guide SC23-0116
- VTAM Diagnosis Reference LY30-5582
- VTAM Data Areas (VM) LY30-5583
- RSCS Networking Version 2 General Information GH24-5055
- RSCS Networking Version 2 Planning and Installation SH24-5057
- RSCS Networking Version 2 Operation and Use SH24-5058
- RSCS Networking Version 2 Diagnosis Reference LY24-5228
- RSCS Networking Version 2 Ref. Summary SX24-5135
- VM/Pass-Through Facility General Information GC24-5206
- VM/Pass-Through Facility Guide and Reference SC24-5208
- VM/Pass-Through Facility Logic LY24-5208

Index

International Business
Machines Corporation
P.O. Box 6
Endicott, New York 13760

File No. S370/4300-39
Printed in U.S.A.

LY20-0893-4

IBM®

VM/SP System Logic and Problem Determination Guide
Volume 2 (CMS)
Order No. LY20-0893-4

READER'S COMMENT FORM

Is there anything you especially like or dislike about this book? Feel free to comment on specific errors or omissions, accuracy, organization, or completeness of this book.

If you use this form to comment on the online HELP facility, please copy the top line of the HELP screen.

______________ ______________ ________ Help Information line ____ of ____

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you, and all such information will be considered nonconfidential.

Note: Do not use this form to report system problems or to request copies of publications. Instead, contact your IBM representative or the IBM branch office serving you.

Would you like a reply? ___YES ___NO

Please print your name, company name, and address:

____

____

____

____

IBM Branch Office serving you: ____________________________________

Thank you for your cooperation. You can either mail this form directly to us or give this form to an IBM representative who will forward it to us.

LY20-0893-4
File No. S370/4300-39

Reader's Comment Form

CUT
OR
FOLD
ALONG
LINE

Fold and tape — Please Do Not Staple — Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE:

INTERNATIONAL BUSINESS MACHINES CORPORATION
DEPARTMENT G60
PO BOX 6
ENDICOTT NY 13760-9987

Fold and tape — Please Do Not Staple — Fold and tape

International Business
Machines Corporation
P.O. Box 6
Endicott, New York 13760

File No. S370/4300-39
Printed in U.S.A.

LY20-0893-4

LY20-0893-04