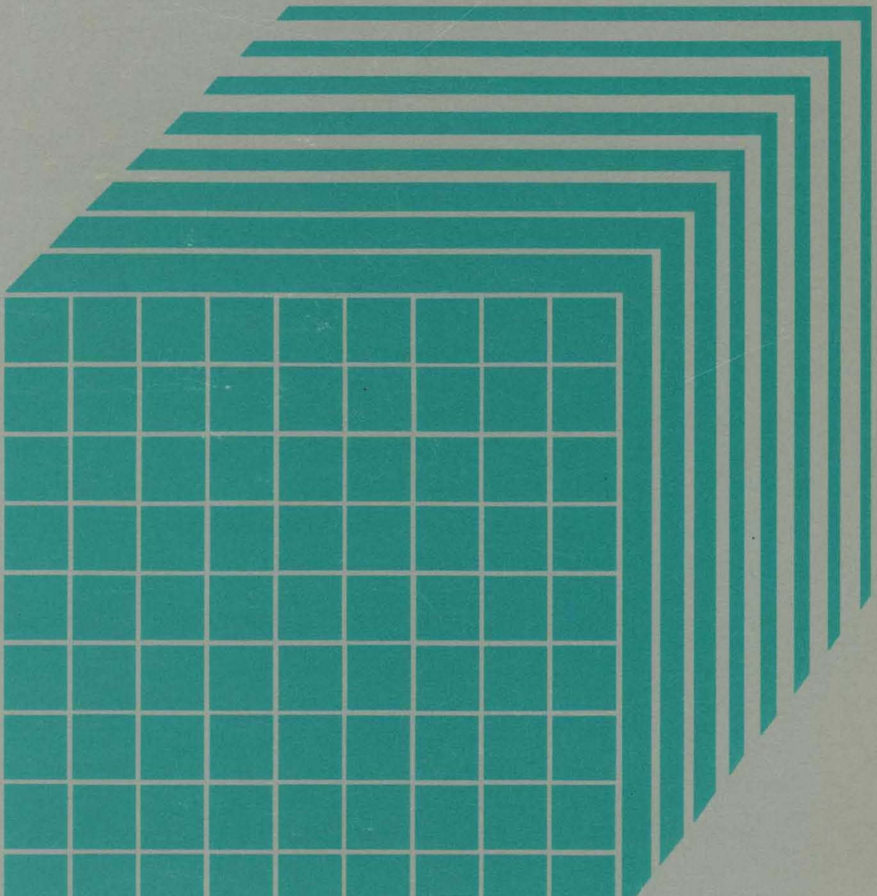
IBM

Virtual Machine/ System Product

CMS for System Programming

Release 5

SC24-5286-0



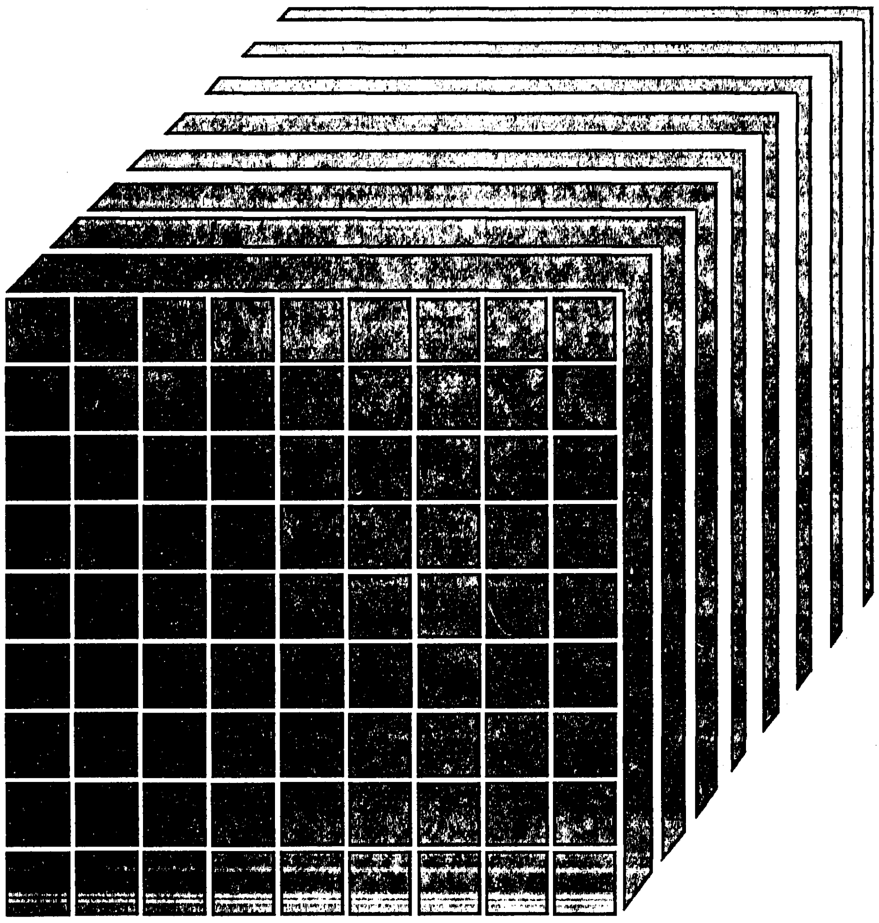
IBM

Virtual Machine/ System Product

CMS for System Programming

Release 5

SC24-5286-0



First Edition (December 1986)

This edition, SC24-5286-0, applies to Release 5 of IBM Virtual Machine/System Product (VM/SP) (program number 5664-167) unless otherwise indicated in new editions or Technical Newsletters. It contains material formerly found in the *VM/SP System Programmer's Guide*, SC19-6203 and the *VM/SP CMS User's Guide*, SC19-6210. 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 detailed list of changes, see page "Summary of Changes" on page 379.

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

Ordering Publications

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Dept. G60, P.O. Box 6, Endicott, New York, 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 publication describes reference information about the functions of the Conversational Monitor System (CMS) component of VM/SP. The information is intended for system programmers, system analysts, and programming personnel. Knowledge of Basic Assembler language and experience with programming concepts and techniques are prerequisites to using this publication.

This publication is one of a set of reference manuals for VM/SP system programmers. The other books in the set include:

- *VM/SP CP for System Programming*
- *VM/SP Group Control System Command and Macro Reference*
- *VM/SP Transparent Services Access Facility Reference*
- *VM System Facilities for Programming*
- *VM Diagnosis Guide*

The order numbers for these books and related publication can be found under “Bibliography” in the back of this manual.

Some of the topics discussed in this publication are:

- Processing abends
- Handling interrupts
- Using storage
- Developing and executing programs
- Linkage conventions
- Updating programs
- Developing commands and messages
- Developing OS programs
- Developing VSE programs
- Using VSAM functions
- Tailoring your CMS system
- Using the batch facility
- Using auxiliary directories
- Understanding assembler virtual storage requirements
- CMS macro library

After the appendices, the following sections are available to help you use this manual more easily:

- “Summary of Changes”
- “Glossary of Terms and Abbreviations”
- “Bibliography”
- “Index”

Contents

Figures

Chapter 1. Introducing 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 system. 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. This makes debugging 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 user can enter commands on the terminal that will alter the program. This is the most common method used to debug programs that run in CMS.

CMS, operating with the VM/SP Control Program (CP), 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. The CMS commands that are useful in debugging are discussed in the *VM Diagnosis Guide* manual. For detailed information on all other CMS commands, refer to the *VM/SP CMS Command Reference.*

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 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:

- A filename
- A filetype implying specific file characteristics to the CMS file management routines.
- A filemode describing the location and access mode of the file.

User files can be created and changed directly from the terminal with the VM/SP System Product Editor (XEDIT). XEDIT provides extensive context editing services. File characteristics such as record length, record format, tab locations, and serialization options can be specified. See *VM/SP System Product Editor Command and Macro Reference* for more information on XEDIT.

The size of user files is determined by the blocksize (BLKSIZE). 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. When a blocksize of 512, 1024, 2048, or 4096 bytes is specified, a single user file is limited to a maximum of 2^{31}-1 CMS records 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 on a disk is limited by the available disk space and the size of the user's files.

When you access a read-only disk, a hyperblock mapping table (HYPMAP) is built. When you access a read/write disk, a hash table complex (HASHTAB) is built. (For further details on HYPMAP and HASHTAB, see the *VM/SP Data Areas and Control Block Logic Volume 2 (CMS)* manual.) These two tables decrease the paging overhead when searching for files. However, the hyperblock mapping table is not built if the hyperblocks for the disk do not span three or more pages. The hash table is not built if the hyperblocks for the disk do not span two or more pages.

CMS automatically allocates compiler work files at the beginning of command execution on whichever active disk has the greatest amount of available space, and 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 are identified by combining the input filename with the filetypes TEXT and LISTING. These disk locations may be overridden by the user.

Virtual disks may be shared by CMS users. This capability is provided by VM/SP to all virtual machines. 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 using any record length and blocksize; the record format can be fixed, variable, or undefined.

Preferred Filetypes

CMS has a list of preferred filetypes. This list consists of filetypes that are frequently searched for, but rarely found on your disk. The list of preferred filetypes is as follows:

```
EXEC
MODULE
CMSUT1
AUTOSAVE
XEDTEMP
XEDIT
SYSUT1
TEXT
```

The active disk table (ADT) contains a byte signalling which preferred filetypes are on the disk. Before scanning the file management tables for a file, this byte is examined to see if any files of the desired type are present on the disk. This process avoids searching for a file that is not on the disk; therefore, improving system performance.

For example, if you are looking for a file with one of the preferred filetypes and the byte in the ADT indicates that the filetype is not on the disk, then you will avoid searching the disk for the file.

Performance may be improved by keeping preferred filetypes together on separate disks.

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. The simulation restrictions concerning what types of OS object programs can be executed under CMS are primarily related to the OS/PCP, MFT, and MVT Indexed Sequential Access Method (ISAM) and the telecommunications access methods. Functions related to multitasking in OS and VSE are ignored by CMS. For more information, see "OS Data Management Simulation" on page 188 and "Chapter 9. Developing VSE Programs under CMS" on page 207.

Chapter 2. Processing Abends

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, 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” attributes.

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.

Chapter 3. Handling Interrupts in CMS

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

SVC Interrupts

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 purpose 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, then 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.

See pages 48 and 52 for more details on SVC 202 and SVC 203.

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 the standard system table (contained in DMSSVT) of OS calls is searched 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, then the supervisor call is treated as an abend call.

The DMSHDS initialization program sets up the user-defined SVC table. The user can provide his own SVC routines by using the HNDSVC macro.

Input/Output Interrupts

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 scans console facility 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 facility 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 nonterminal (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 interrupting 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 that 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 Interrupts

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 device end.

Reader/Punch/Printer Interrupts

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. Then 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 Interrupts

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 specified in the user device table. Otherwise, the processing program regains control directly.

To handle unsolicited device interrupts, you may specify the EXIT parameter for the OPEN request of the CONSOLE macro instruction. If you specify this parameter, do NOT define an interruption routine via the HNDINT macro for the same device. Use of the CONSOLE macro with the use of HNDINT should be mutually exclusive. 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.

The console facility 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, the console facility helps CMS keep track of what application is doing I/O or what application handled interrupts last.

The CONSOLE macro supersedes an HNDINT routine when the interrupt is solicited. In most cases, a CONSOLE WAIT and CONSOLE READ can be issued instead of coding an HNDINT routine to handle all interrupts. Therefore, if you want to perform I/O to a 3270 device, you should use the CONSOLE macro instead of the HNDINT macro.

Program Interrupts

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 the 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 Interrupts

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 Interrupts

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 to continue.

Chapter 4. Using Storage

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.

Structure of CMS Storage

Figures 1, 2, and 3 on pages 18, 19, and 20 describe 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 following uses:

- **DMSNUC (X'00000' to ANUCEND).** This is the nucleus constant area. It contains system control blocks, 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. See "User and Transient Program Areas" on page 23 for more details on the transient program area.
- **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-disk. 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 lower 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. See "User and Transient Program Areas" on page 23 for details on the user program area.

- **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 (NUCALPHA to NUCOMEGA).** 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 1 on page 18), 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 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 ending location of the CMS nucleus (refer to Figure 2 on page 19 and Figure 3 on page 20), 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 insufficient room to place the loader table above the S- and Y-STAT, the loader table is placed below the nucleus. Any leftover free space above the S- and Y-STAT (NUCOMEGA) is placed on the high DMSFREE chain.

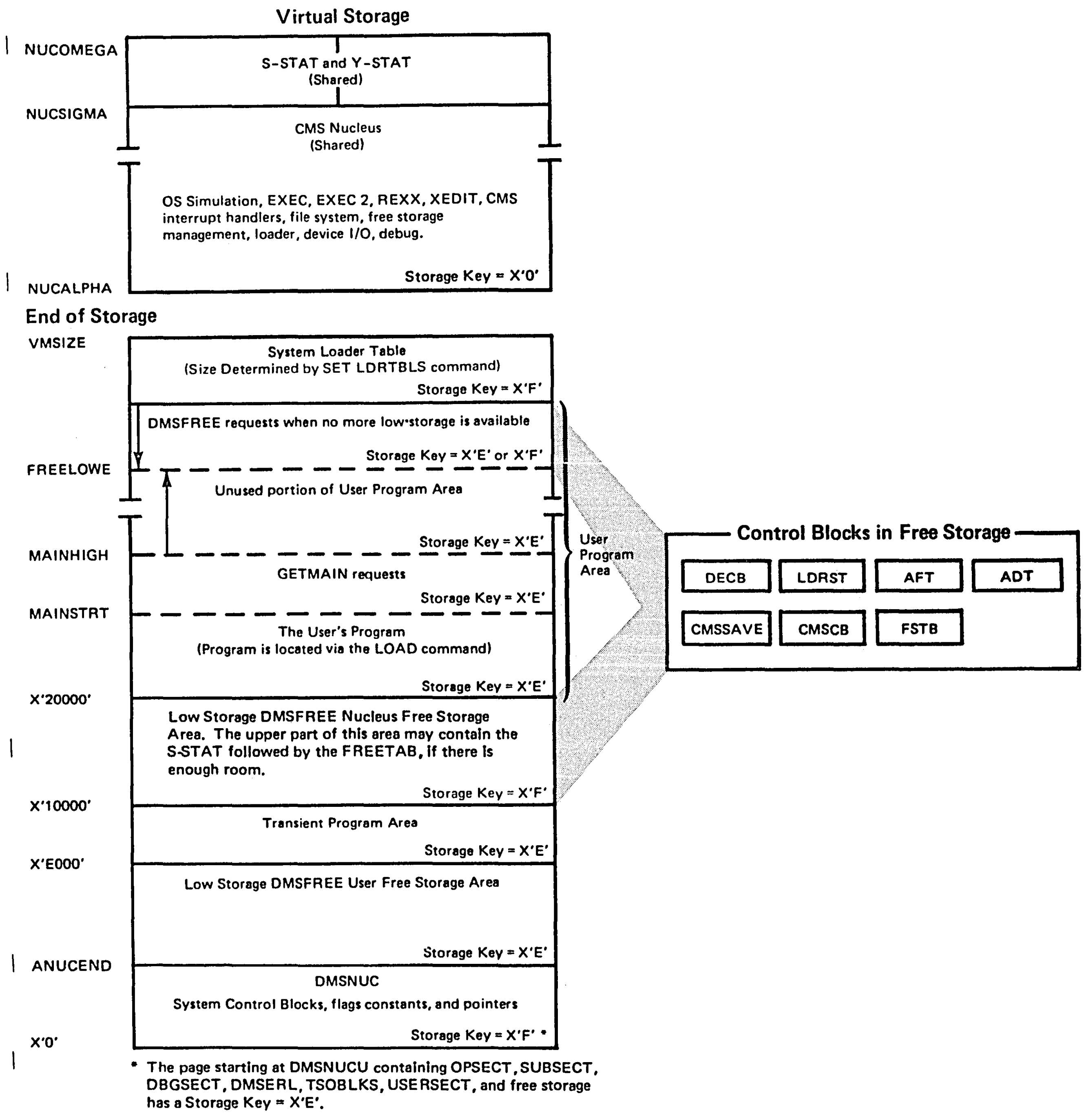

Virtual Storage
NUCOMEGA
S-STAT and Y-STAT
(Shared)
NUCSIGMA
CMS Nucleus
(Shared)
OS Simulation, EXEC, EXEC 2, REXX, XEDIT, CMS interrupt handlers, file system, free storage management, loader, device I/O, debug.
Storage Key = X'0'
NUCALPHA
End of Storage
VMSIZE
System Loader Table
(Size Determined by 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
User Program Area
GETMAIN requests
Storage Key = X'E'
MAINSTRT
The User's Program
(Program is located via the LOAD command)
Storage Key = X'E'
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'
* The page starting at DMSNUCU containing OPSECT, SUBSECT, DBGSECT, DMSERL, TSOBLKS, USERSECT, and free storage has a Storage Key = X'E'.
Control Blocks in Free Storage
DECB
LDRST
AFT
ADT
CMSSAVE
CMSCB
FSTB

Figure 1. 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.

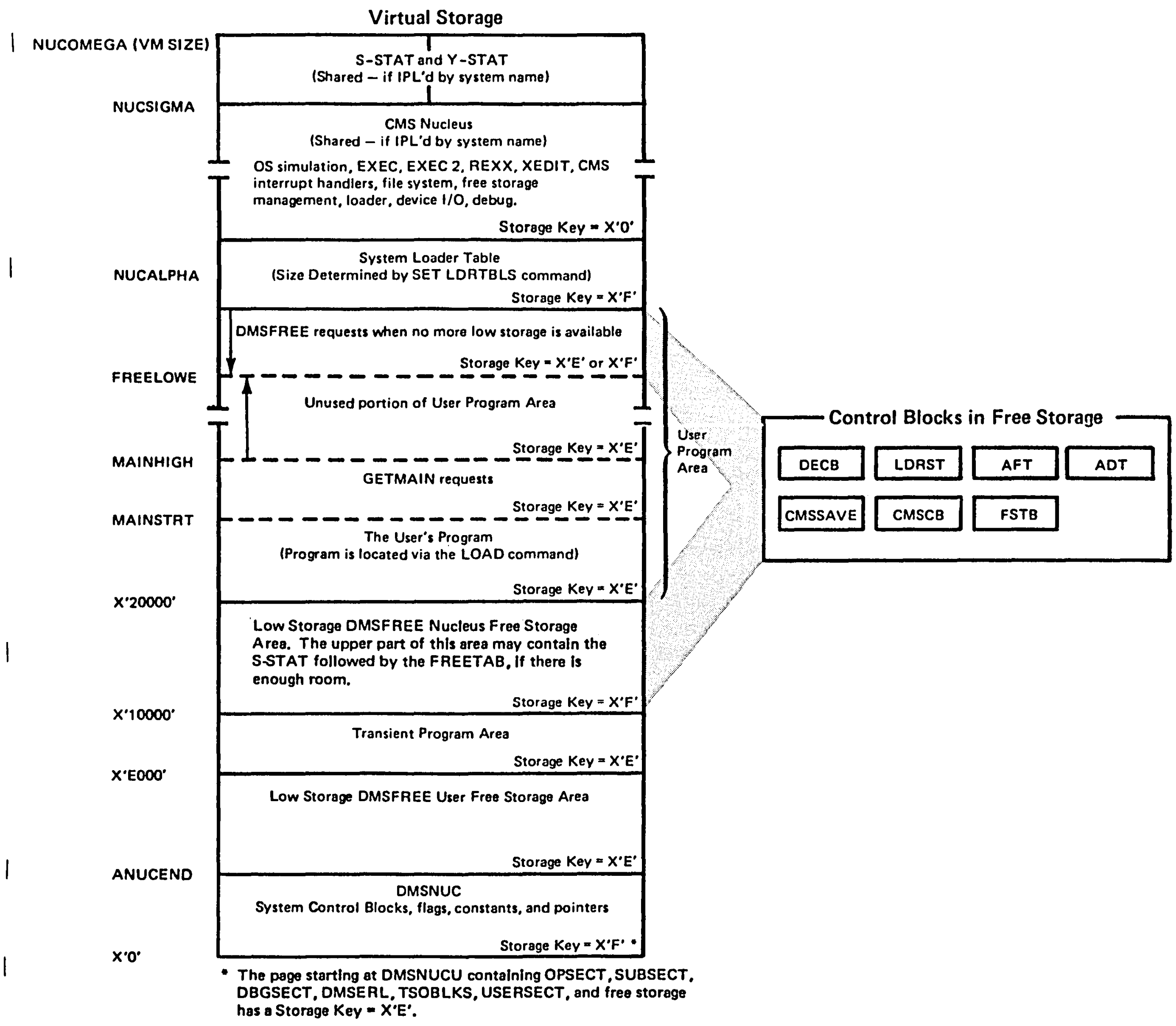
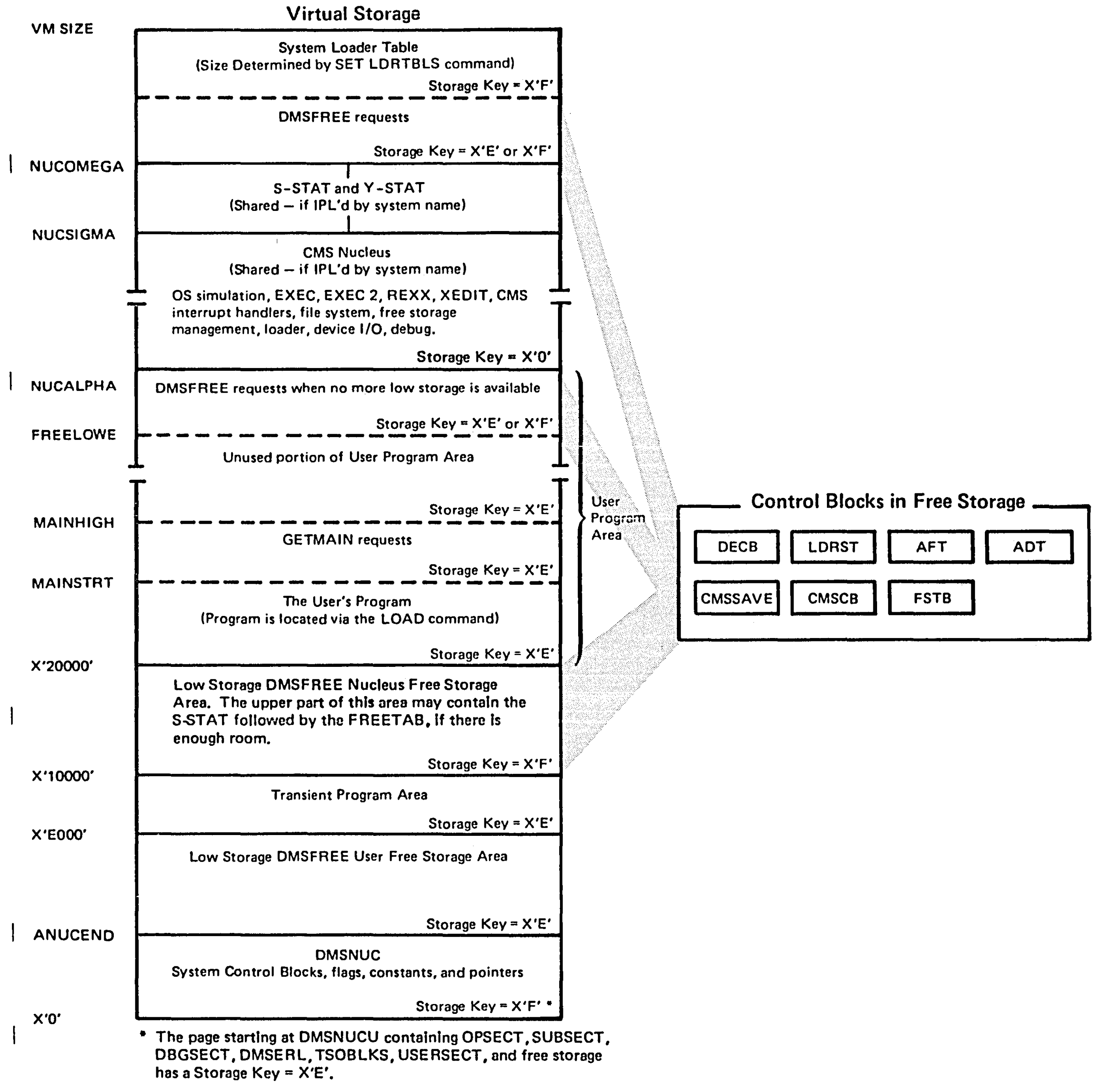

Virtual Storage
NUCOMEGA (VM SIZE)
S-STAT and Y-STAT
(Shared — if IPL'd by system name)
NUCSIGMA
CMS Nucleus
(Shared — if IPL'd by system name)
OS simulation, EXEC, EXEC 2, REXX, XEDIT, CMS interrupt handlers, file system, free storage management, loader, device I/O, debug.
Storage Key = X'0'
NUCALPHA
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'
* The page starting at DMSNUCU containing OPSECT, SUBSECT, DBGSECT, DMSERL, TSOBLKS, USERSECT, and free storage has a Storage Key = X'E'.
Control Blocks in Free Storage
DECB
LDRST
AFT
ADT
CMSSAVE
CMSCB
FSTB

Figure 2. 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 the case 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
VM SIZE
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 3. **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 the case where there is sufficient 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.

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. Figure 4 describes the devices supported by CMS.

Virtual IBM Device Type	Virtual Address*	Symbolic Name (default)	Device Use
3210, 3215, 1052, 3066, 3270	cuu[1]	CON1	System console
2314, 2319, 3310, 3330, 3340, 3350, 3370, 3375, 3380	190	DSK0	CMS System disk (read-only)
	191[2]	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	DSK8	Minidisk (user files)
	cuu	DSK9	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, 4245, 4248, 3289-4	00E	PRN1	Line printer
2401, 2402, 2403, 2415, 2420, 3410, 3411, 3420, 3430, 3480, 8809, 3422	180 - 187, 288 - 28F	TAP0 - TAP7, TAP8 - TAPF	Tape drives

Figure 4. Devices Supported by a CMS Virtual Machine

* The device addresses shown are preassembled into the CMS resident device table. These need only be modified and a new device table made resident to change the addresses.

[1] The virtual address of the system console may be any valid multiplexer address.

[2] 191 is the default user-accessed A-disk unless it is dynamically changed by an ACCESS at CMS initial program load (IPL).

User and Transient Program Areas

Two areas hold programs that are loaded from disk. These areas are called the user program area and the transient program area, as discussed on page 15. (See Figures 1, 2, and 3 on pages 18, 19, and 20 for a description of CMS storage use.) 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. Generally, all user programs and certain system commands are executed in the user program area. Because only one program can be executing in the user program area at any 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 in the user program area.

The transient program area is two pages long, extending 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, a program executing in the transient program area may not invoke the TYPE command.

DMSITS starts the programs to be executed in the user program area enabled for all interrupts, but DMSITS starts the programs to be executed in the transient program area disabled for all interrupts. The individual programs may have to use the SSM (Set System Mask) instruction to change the current status of its system mask.

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 separate 4K pages. It is possible 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 function initializes pointers used by CMS for simulation of OS GETMAIN/FREEMAIN storage management. In the usual CMS environment, that is, when execution is initiated by the LOAD and START commands, CMS calls the STRINIT macro as 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. In addition, the STRINIT function makes any pages that were allocated by GETMAIN available to be released by the CMS page manager.

The STRINIT Macro

The format of the STRINIT macro is:

[*label*]	**STRINIT**	$\left[\textbf{TYPCALL}=\left\{\begin{array}{l}\underline{\textbf{SVC}}\\\textbf{BALR}\end{array}\right\}\right]$

where:

label
is any valid assembler language label.

$\textbf{TYPCALL}=\left\{\begin{array}{l}\underline{\textbf{SVC}}\\\textbf{BALR}\end{array}\right\}$
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 (see Notes below). 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.

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 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. Refer to Figures 1, 2, and 3 on pages 18, 19, and 20 for a description of CMS virtual storage usage.

Notes:

1. *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. This may cause a storage management failure.*
2. *A high level language may issue a STRINIT. In this case, a user should not issue an additional STRINIT.*

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:

1. A specific block of such storage may be released by means of the FREEMAIN macro instruction.
2. 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 pointers are reset.
3. 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:

[*label*]	**DMSFREE**	**DWORDS=** $\left\{\begin{matrix} n \\ \mathbf{(0)} \end{matrix}\right\}$ $\left[\textbf{,MIN=}\left\{\begin{matrix} n \\ \mathbf{(1)} \end{matrix}\right\}\right]$ $\left[\textbf{,TYPE=}\left\{\begin{matrix} \underline{\textbf{USER}} \\ \textbf{NUCLEUS} \end{matrix}\right\}\right]$ $\left[\textbf{,ERR=}\left\{\begin{matrix} laddr \\ * \end{matrix}\right\}\right]$ $\left[\textbf{,AREA=}\left\{\begin{matrix} \textbf{LOW} \\ \textbf{HIGH} \end{matrix}\right\}\right]$ $\left[\textbf{,TYPCALL=}\left\{\begin{matrix} \underline{\textbf{SVC}} \\ \textbf{BALR} \end{matrix}\right\}\right]$

where:

label
is any valid assembler language label.

DWORDS= $\left\{\begin{matrix} n \\ \mathbf{(0)} \end{matrix}\right\}$
is the number of doublewords of free storage requested. DWORDS=*n* specifies the number of doublewords directly and 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= $\left\{\begin{matrix} n \\ \mathbf{(1)} \end{matrix}\right\}$
indicates a variable request for free storage. If the exact number of doublewords indicated by DWORDS operand is not available, then the largest block of storage greater than or equal to the minimum is requested. 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 by 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 wherever 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 indicates 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 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 always in multiples of 4K bytes so the pointer is always on a 4K boundary.

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. Figures 1, 2, and 3 on pages 18, 19, and 20 show 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:

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:

0(0)	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.			
4(4)	NUM — the number of elements on the chain.			
8(8)	MAX — a value equal to or greater than the size of the largest element on the chain.			
12(C)	FLAGS-Flag byte	SKEY-Storage key	TCODE-FREETAB code	Unused

where:

POINTER
points to the first FBD (file 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 zeroes.

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 the 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.

FLUN (X'10') -- Nucleus chain. Set for both the low-storage and high-storage 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 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 try to satisfy the request. As soon as one of the following steps succeeds, the 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 requested 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, DMSFREE 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:

1. A specific block of such storage may be released by means of the DMSFRET macro instruction.

2. Whenever any user routine or 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. It 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 format of the DMSFRET macro is:

[*label*]	**DMSFRET**	**DWORDS** = $\left\{\begin{matrix} n \\ \mathbf{(0)} \end{matrix}\right\}$ **,LOC** = $\left\{\begin{matrix} laddr \\ \mathbf{(1)} \end{matrix}\right\}$ $\left[\textbf{,ERR} = \left\{\begin{matrix} laddr \\ * \end{matrix}\right\}\right]$ $\left[\textbf{,TYPCALL} = \left\{\begin{matrix} \underline{\textbf{SVC}} \\ \textbf{BALR} \end{matrix}\right\}\right]$

where:

label
is any valid assembler language label.

DWORDS = $\left\{\begin{matrix} n \\ \mathbf{(0)} \end{matrix}\right\}$
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 = $\left\{\begin{array}{l}\textit{laddr}\\ *\end{array}\right\}$

is the return address if any 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 the asterisk (*) is specified, the error return address is the same as a normal return address. There is no default for this operand. If it is omitted and an error occurs, the system abends.

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. When FREELOWE is adjusted upward, the corresponding pages are released by issuing a DIAGNOSE code X'10' instruction to CP. The CMS page manager is notified of any completely unallocated 4K pages.

CMS Page Management

The CMS page manager (DMSPAG) controls the release of de-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 instruction to request certain free storage management services.

The DMSFRES Macro

The format of the DMSFRES macro is:

[*label*]	**DMSFRES**	**INIT1**	
		INIT2	
		CHECK	$\left[\text{,TYPCALL} = \left\{ \begin{array}{l} \underline{\text{SVC}} \\ \text{BALR} \end{array} \right\} \right]$
		CKON	
		CKOFF	
		UREC	
		CALOC	

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 are subject to the following restraints until the second free storage management initialization routine has been invoked:

- All requests for USER type storage are changed to requests for NUCLEUS type 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, since 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 just 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 and the PAGETAB page management table. These tables each contain one byte for each 4K page of virtual storage. Therefore, the tables cannot be allocated until the size of virtual storage is known.

  They are allocated in the nucleus low free storage area, if there is enough room available. If not, then they are allocated in the higher free storage area. For a 256K virtual machine, FREETAB and PAGETAB each contain 64 bytes; for a 16 million byte machine, they each contain 4096 bytes.
- The FREETAB and PAGETAB tables are initialized, 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 or not any free storage pointers have been destroyed. The 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 or not the abend recovery has been successful.

TYPCALL = $\left\{\begin{array}{l}\underline{\textbf{SVC}}\\ \textbf{BALR}\end{array}\right\}$

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).

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.

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** $\left\{\begin{array}{l}\textbf{ON}\\\underline{\textbf{OFF}}\end{array}\right\}$

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:

[*label*]	**DMSKEY**	**{NUCLEUS [,NOSTACK] | USER [,NOSTACK] | LASTUSER [,NOSTACK] | RESET}**

where:

label
is any valid assembler language label.

NUCLEUS
causes the nucleus storage protection key to be placed ın 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 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 OI,OSSFLAGS,COMPSWT
```

causes the OI instruction to be executed with a zero protect key in the PSW. This sequence turns on the COMPSWT flag in the nucleus. It is reset with

```
DMSEXS NI,OSSFLAGS,255-COMPSWT
```

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.

Program Linkage (SVC Handling)

Program linkages, in CMS, are made by a supervisor call instruction, SVC 202. 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. The DMSITS module is entered whenever a supervisor call is executed.

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

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

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

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

Register Usage

When calling a CMS routine, 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) was 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.

When a command or routine is called by SVC 202, the registers contain the following information:

Register	Contents
0	Points to an extended PLIST if the command is: • called from the terminal, • called from a REXX program, • called from an EXEC 2 EXEC, or • an Enhanced Connectivity Facilities on VM/SP call (see SENDREQ in the *VM/SP IBM Programmer's Guide to the Server-Requester Programming Interface for VM/SP*, SC24-5291). The EPLIST contains addresses referring to the extended command as it was initially entered by the user.
1	Points to a parameter list of successive doublewords. The first entry in the list is the name of the called routine or program. Any successive doublewords may contain arguments passed to the program.
13	Contains the address of a 24-fullword save area, which you can use to save your caller's registers. This save area satisfies standard OS and DOS linkage conventions. You do not need to use it in CMS, since the SVC routines save the registers.
14	Contains the return address of the SVC handling routines. You must return control to this address when you exit from your program.
12 and 15	Contain your program's entry point address. You can use this address to establish immediate addressability in your program. Most CMS routines use R12 as a base register. You should not use R15 as a base address, since all CMS SVCs use it to communicate with your programs. 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.

Figure 5 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 5. 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 6 show 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 6. 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.*

Parameter Lists

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 being called. These arguments must be "tokenized." Every parenthesis is considered an individual argument, and each argument may have a maximum length of eight characters. However, no error condition results. R1 contains the address of this parameter list.

CMS commands look for a token of eight X'FF's to find the end of the PLIST.

See page 51 for an example of a tokenized PLIST.

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.

The First Form of the Extended PLIST: 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.

See page 51 for an example of an extended PLIST.

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 the Extended PLIST: The second form of an extended PLIST is used by Enhanced Connectivity Facilities on VM/SP (see SENDREQ in the *VM/SP IBM Programmer's Guide to the Server-Requester Programming Interface for VM/SP*, SC24-5291). The second form provides a way for a routine to:

- Pass up to 64K-1 bytes of arbitrary data and 32K-5 bytes of parameters to another routine

- Receive up to 64K-1 bytes of arbitrary data and 32K-5 parameters from another routine.

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

```
(a) DC A(commandname)
(b) DC F      (reserved)
(c) DC F      (reserved)
(d) DC 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 connectivity program 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. 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 IBM Programmer's Guide to the Server-Requester Programming Interface for VM/SP*, SC24-5291.

Common SVC Calls

SVC conventions are important to any discussion of CMS because the system is driven by SVCs (supervisor calls). SVCs 202 and 203 are the most common CMS SVCs.

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)
```

First, load the address of the parameter list into R1 and then issue an SVC. The "DC AL4(address)" instruction following the SVC 202 is optional and may be omitted if you do not expect any errors to occur in the routine or command being called.

If the DC statement is included and the return code (R15) contains a nonzero value after returning from the SVC call, control passes to the address specified in the DC unless the address is equal to 1. In the above example, control would go to the instruction at the label ERRADD.

If the address is 1, return is made to the instruction following the "DC AL4(1)" instruction. DMSITS determines whether this DC was inserted by examining the byte following the SVC call. If the byte is nonzero, the statement following the SVC 202 is an instruction. If the byte is zero, the statement following the SVC 202 is either "DC AL4(address)" or "DC AL4(1)".

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

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

Whenever an SVC 202 is issued, the contents of general purpose registers 0 and 1 (R0 and R1) are passed to the called routine. R1 points to an 8-character string, which may be the start of a tokenized PLIST. This character string contains 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. The SVC handler only examines the name and the high-order byte of R1.

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

When a program gets control, it checks the value of the the high-order byte of R1 to determine what environment (EXEC, command line, etc.) it was called from and if an extended PLIST is available. Then the program may take appropriate action. CMS only places these values in the high-order byte for the convenience of the program.

The following values may be found in the high-order byte of register 1:

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 Enhanced Connectivity Facilities on VM/SP call (see SENDREQ in the *VM/SP IBM 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 46. (The SVC handler translates the value X'03' to X'01' before entering the called program.)	Yes
X'02'	See "Dynamic Linkage/SUBCOM" on page 59.	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 EXEC when "ADDRESS CMS" is specified.	Yes

Figure 7 (Part 1 of 2). SVC 202 High-Order Byte Values of Register 1

Value	Meaning	Extended PLIST Pointer in Register 0?
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 an 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 7 (Part 2 of 2). SVC 202 High-Order Byte Values of Register 1

Some CMS commands work differently when called from different environments. An assembler language program can simulate the various environments (listed in Figure 7 under "Meaning") by using the appropriate high-order byte.

For example, to call the ERASE command from an assembler program and to suppress error messages, the program uses a high-order byte of X'0E'. This simulates a call from a CMS EXEC with "&CONTROL NOMSG" in effect.

Some CMS commands can take advantage of an extended PLIST if it is supplied. For example, the FILEDEF command uses the extended parameter list when processing the DSN qual1[.qual2...] parameter. The following program shows how to set up an extended parameter list and call FILEDEF. The high-order byte, X'01', in the program example simulates a call from an EXEC2 EXEC or the System Product Interpreter when "ADDRESS COMMAND" is specified.

```
SAMPLE   CSECT
*
*        ISSUE 'FILEDEF SYSIN DISK A A A DSN G.TEMP.DATA.LIBRARY'
*
         REGEQU
         USING *,R12
         LR    R12,R15
         LA    R0,EPLIST
         LA    R1,PLIST
         ICM   R1,B'1000',=X'01'
         SVC   202
         DC    AL4(1)
         BR    R14
         DS    0F
PLIST    DC    CL8'FILEDEF '
         DC    CL8'SYSIN'
         DC    CL8'DISK'
         DC    CL8'A'
         DC    CL8'A'
         DC    CL8'A'
         DC    CL8'DSN'
         DC    CL8'G.TEMP.D'  NOTICE THAT THIS IS TRUNCATED
*                             BUT FILEDEF WILL USE THE
*                             EXTENDED PARAMETER LIST
*                             BELOW.
         DC    2F'-1'
EPLIST   DC    A(COMVERB)
         DC    A(BEGARGS)
         DC    A(ENDARGS)
         DC    A(0)
COMVERB  DC    C'FILEDEF '
BEGARGS  EQU   *
         DC    C'DISK A A A DSN G.TEMP.DATASET.LIBRARY'
ENDARGS  EQU   *              ENDARG POINTS ONE CHARACTER
*                             PAST THE END OF THE
*                             PARAMETER LIST.
         END
```

Refer to page 43 for a description of the contents of R12, R13, R14, and R15.

SVC 202 Return Codes: On return from SVC processing, register 15 contains one of the following return codes:

0 No errors occurred.

-1 A CP command with this name was not found.

-2 An attempt was made to execute a CMS command while in CMS subset mode. This would have caused the module to be loaded in the user area.

-3 A CMS command issued from EXEC was not found with this name, or an invalid function occurred when the SET or QUERY command was issued from EXEC with IMPCP active.

-4 The LOADMOD failed.

-5 A LOADMOD was issued in the wrong environment (for example, the module was generated by the GENMOD command with the OS option, and LOADMOD was attempted with DOS=ON specified).

SVC 203

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

A typical calling 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 a 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 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 handled in the same way as the 8-byte name passed in the parameter list to an 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 routine called.

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 X'3' and X'259' to be identical and handles them the same as X'-3' and X'-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 and VSE Macro Simulation SVC Calls

CMS supports selected SVC calls generated by OS and VSE macros by simulating the effect of these macro calls. DMSITS is the initial SVC interrupt handler. If the SET DOS command has been issued, a flag in NUCON indicates that VSE macro simulation is to be used. Control is then passed to DMSDOS. Otherwise, OS macro simulation is assumed and DMSITS passes control to the appropriate OS simulation routine.

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.

Invalid SVC Calls

There are several types of invalid SVC calls recognized by DMSITS.

1. 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 on your terminal, and control is returned directly to the caller.

2. 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 as it handles an error return from a legitimate SVC routine. You receive an error code of -3.

3. Invalid SVC 203 code. If an invalid code follows SVC 203 inline, an error message is displayed on your terminal 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. Figure 9 on page 58 and the following list describe the search order.

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. Figure 8 on page 57 and the following list describe the search order.

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

   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.

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

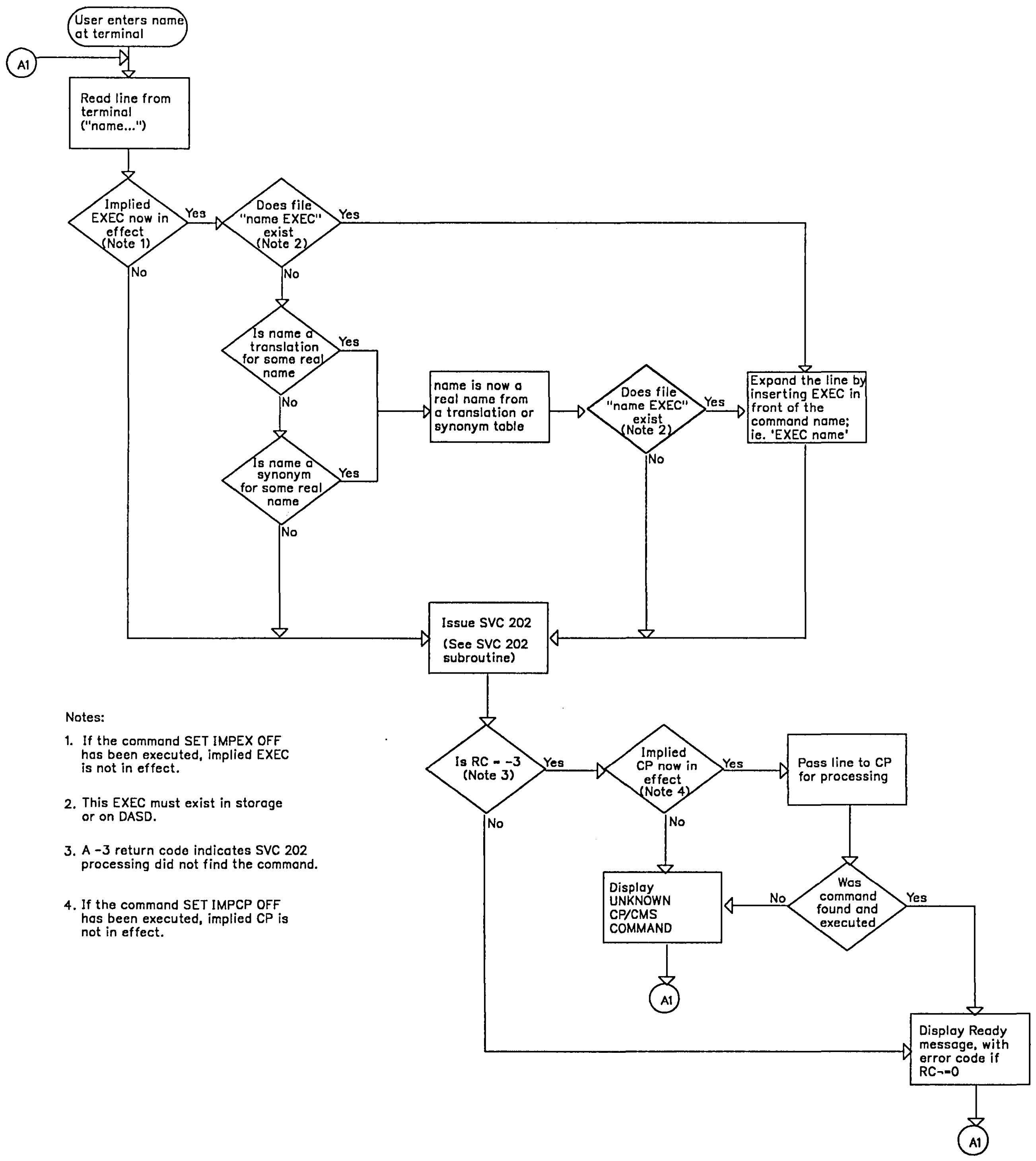
DMSITS performs the search for SVC 202 as described above in "SVC 202 Entered from a Program."

4. DMSINT searches for a CP command with the specified name, using the CP DIAGNOSE instruction.[2]

5. If all of these searches fail, DMSINT displays the error message:

```
Unknown CP/CMS Command
```

[2] 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 8. CMS Command Processing

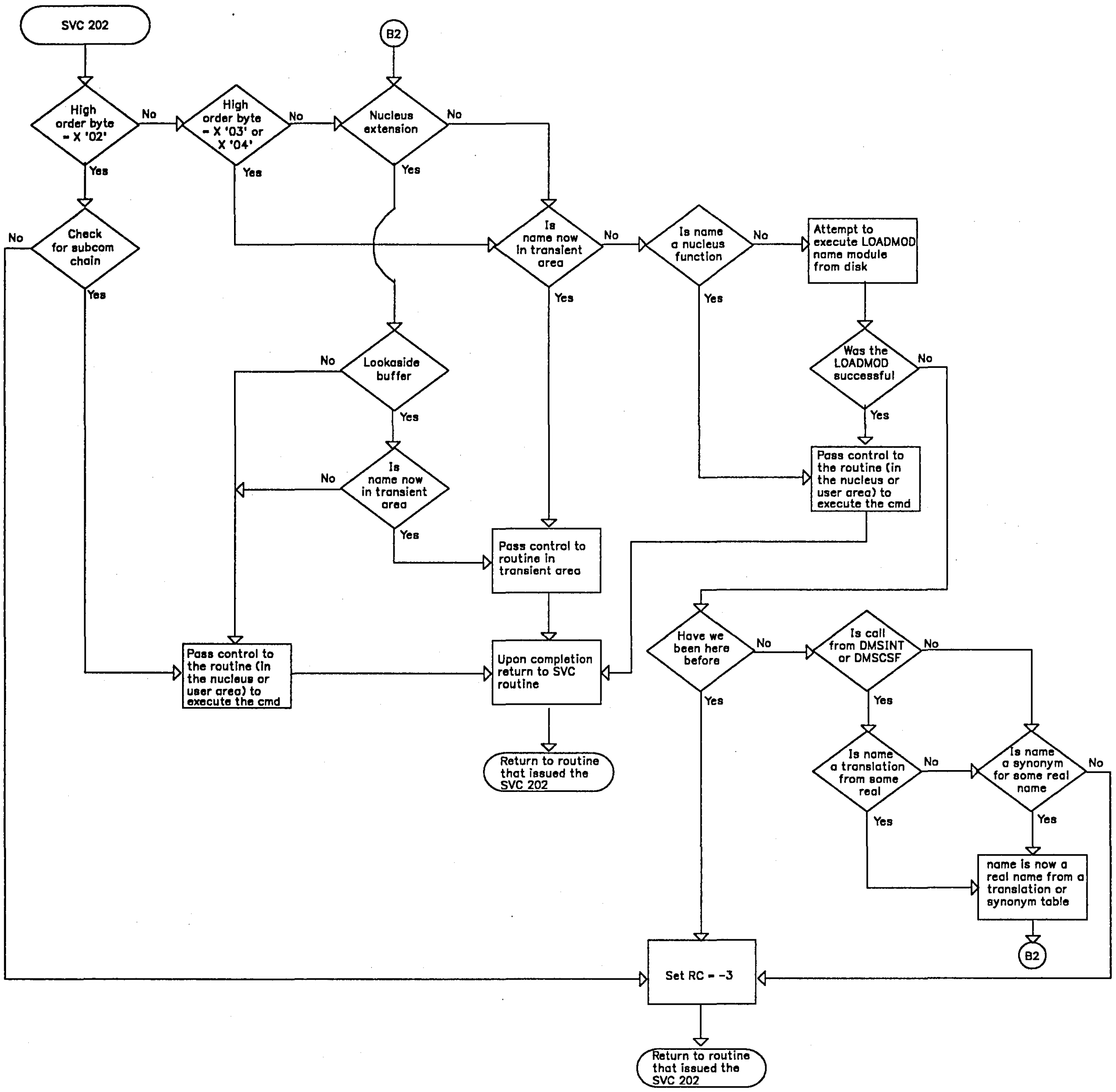

SVC 202
High order byte = X '02'
No
Yes
High order byte = X '03' or X '04'
No
Yes
B2
Nucleus extension
No
Yes
Check for subcom chain
No
Yes
Is name now in transient area
No
Yes
Is name a nucleus function
No
Yes
Attempt to execute LOADMOD name module from disk
Was the LOADMOD successful
No
Yes
Lookaside 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
Upon completion return to SVC routine
Return to routine that issued the SVC 202
Pass control to the routine (in the nucleus or user area) to execute the cmd
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 9. SVC 202 Processing

Command Search Function

SUBCOM provides a function that lets you invoke a command (from a program) that is resolved according to the CMS command search hierarchy. That is, the command is resolved just as though the command was entered from the terminal. This SUBCOM function is named CMS. This command search function checks the IMPEX and IMPCP settings of CMS SET.

The CMS SUBCOM function is defined during system initialization at IPL and remains defined during the entire CMS session.

To pass a command to the CMS SUBCOM function, the user should define PLISTs as follows:

```
PLIST      DS     0F
           DC     CL8'CMS'
EXPLIST    DS     0F
           DC     A(PLIST)
           DC     A(BEGARGS)
           DC     A(ENDARGS)
           DC     A(0)

BEGARGS    DS     0F
           DC     C'command to be invoked'
ENDARGS    EQU    *
```

Register 1 must contain the address of PLIST and a high order byte of X'02'. Register 0 must contain the address of the extended PLIST. Having established the PLIST and register information the user issues an SVC 202. The X'02' in the high order byte of register 1 indicates that this is a call to a previously defined SUBCOM.

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.

SUBCOM Function Return Codes

Return codes from the SUBCOM function are:

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.

Returning to the Calling Routine

When the called routine finishes 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, which 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 upon whether the called routine indicated an error return.

For SVC 202 and 203, the called routine places a zero in register 15 indicating a normal return and a nonzero code in register 15 indicating an error return. 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 address to the calling routine, if one was specified. If no error return address was specified, DMSITS abnormally terminates.

For an 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 back by the called routine.

For an 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 an 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 always contains the value that was in register 15 when the called routine returned to DMSITS after it had completed processing.

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 NRMRET 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 upon return. At the completion of each command, all SVC save areas allocated by that command are released.

DMSITS uses the system save area 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 the *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) 8-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 in case of a normal return from the called routine.
ERRET	(Fullword) The address of the calling routine where control is passed in case of an error return from the called routine.
EGPRS	(16 Fullwords, separately labeled EGPR0, EGPR1, EGPR2, EGPR3, ..., EGPR15) The entry registers. 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 where the current call is nested.
USAVEPTR	(Fullword) Pointer to the user save area for this SVC call.

The CMS Subset Environment

When you issue the XEDIT subcommand:

```
cms
```

the editor responds:

```
CMS subset
```

and your virtual machine is in CMS subset mode. When in subset mode, you can issue any valid CMS subset command, that is, a CMS command that is allowed in CMS subset mode. The commands that are not allowed in the CMS subset environment are commands that execute in the user area. You can also issue CP commands. To return to edit mode, you use the special CMS subset command, RETURN. If you enter the Immediate command HX, your editing session terminates abnormally and your virtual machine returns to the CMS environment.

When entering CMS subset mode either for a single command or until the string 'RETURN' is entered, the following processing is done to ensure that the previous environment is preserved. Upon entry to subset, a check is made to determine if this entry would constitute a recursion, if so, return code 1 is returned.

1. STAE, SPIE, and STAX information is saved and then cleared.
2. The OS environment settings are saved and then cleared so that any module that issues an OSRESET based on these flags will not do so.
3. The read and write pointers from any currently opened files are saved.
4. All files are then closed by a 'FINIS * * *', but files with a filemode of 3 are not erased.
5. Any FSTs that were built by a previous call to STATE are saved.

If the entry to subset was just for the execution of a single command, the entry message is suppressed and the next command is executed immediately. But, if the request was to enter CMS subset for an indefinite duration, an announcement of entry to the CMS subset environment is made. This is done so that a strict differentiation from the strict command environment is given.

The principle difference in subset is the restriction that any command executed may not use any storage other than DMSFREE storage and the transient area. This protects programs which may be running in the USER AREA. Also, any ready message issued from subset is in the abbreviated form (i.e. identical to SET RDYMSG SMSG) so that program timing information is not affected for the command currently in progress at the time of subset entry.

Upon termination of CMS subset mode any settings or values that were saved upon entry to subset are restored.

Assembling Programs

To assemble assembler language source programs into object module format, you can use the ASSEMBLE command, and specify assembler options on the command line. For example:

```
assemble myfile (print
```

assembles a source program named MYFILE ASSEMBLE and directs the output listing to the printer. All of the ASSEMBLE command options are listed in the *VM/SP CMS Command Reference.*

When you invoke the ASSEMBLE command specifying a file with the filetype of ASSEMBLE, CMS searches all of your accessed disks, using the standard search order, until it locates the specified file. When the assembler creates its output listing and text deck, it creates files with filetypes of LISTING and TEXT, and writes them onto disk according to the following priorities:

1. If the source file is on a read/write disk, the TEXT and LISTING files are written onto that disk.

2. If the source file is on a read-only extension of a read/write disk, the TEXT and LISTING files are written onto the parent disk.

3. If the source file is on any other read-only disk, the TEXT and LISTING files are written onto the A-disk.

4. If none of the above choices are available, the command is terminated.

In all of the above cases, the TEXT and LISTING files have a filename that is the same as the input ASSEMBLE file.

The input and output files used by the assembler are assigned by FILEDEF commands that CMS issues internally when the assembler is invoked. If you issue a FILEDEF command using one of the assembler ddnames before you issue the ASSEMBLE command, you can override the default file definitions.

The ddname for the source input file (SYSIN) is ASSEMBLE. If you enter:

```
filedef assemble reader
assemble sample
```

then the assembler reads your input file from your card reader and assigns the filename SAMPLE to the output TEXT and LISTING files.

You could assemble a source file directly from an OS disk by entering:

```
filedef assemble disk myfile assemble b4 dsn os source file
assemble myfile
```

In this example, the CMS file identifier MYFILE ASSEMBLE is assigned to the data set OS.SOURCE.FILE and then assembled.

LISTING and TEXT are the ddnames assigned to the SYSPRINT and SYSLIN output of the assembler. You might assign file definitions to override these defaults as follows:

```
filedef listing disk assemble listfile a
filedef text disk assemble textfile a
assemble myfile
```

In this example, output from the assembly of the file, myfile ASSEMBLE, is written to the files, ASSEMBLE LISTFILE and ASSEMBLE TEXTFILE.

The ddnames PUNCH and CMSLIB are used for SYSPUNCH and SYSLIB data sets. PUNCH output is produced when you use the DECK option of the ASSEMBLE command. The default file definition for CMSLIB is the macro library CMSLIB MACLIB, but you must still issue the GLOBAL command if you want to use it.

Executing Programs

After you have assembled or compiled a source program, you can execute the TEXT files that were produced by the assembly or compilation. You may not, however, be able to execute all your OS programs directly in CMS. There are a number of execution-time restrictions placed on your virtual machine by VM/SP. You cannot execute a program that uses:

- Multitasking
- More than one partition
- Teleprocessing
- ISAM macros to read or write files
- The PSW EC mode bit

The above is only a partial list of restrictions you might be concerned with. For a complete list of restrictions, see the *VM/SP Planning Guide and Reference.*

Executing TEXT Files

TEXT files, in CMS, are relocatable and can be executed simply by loading them into virtual storage with the LOAD command and using the START command to begin execution. For example, if you have assembled a source program named CREATE, you have a file named CREATE TEXT. You can issue the command:

```
load create
```

that loads the relocatable object file into storage. Then, to execute it, you can issue the START command:

```
start
```

In the case of a simple program, as in the above example, you can load and begin execution with a single command line, using the START option of the LOAD command:

```
load create (start
```

When you issue the START command or LOAD command with the START option, control is passed to the first entry point in your program. If you have more than one entry point and you want to begin execution at an entry point other than the first, you can specify the alternate entry point or CSECT name on the START command:

```
start create2
```

When you issue the LOAD command specifying the filename of a TEXT file, CMS searches all of your accessed disks for the specified file.

If your program expects a parameter list to be passed (via register 1), you can specify the arguments on the START command line. If you enter arguments, you must specify the entry point:

```
start * name1
```

When you specify the entry point as an asterisk (*), it indicates that you want to use the default entry point.

Defining Input and Output Files

You can issue the FILEDEF command to define input and output files any time before you begin program execution. You can issue all your file definitions before loading any TEXT files, or you can issue them during the loading process. You can find out what file definitions are currently in effect by issuing the FILEDEF command with no operands. You can also use the FILEDEF operand of the QUERY command.

Resolving External References

The CMS loader loads files into storage as a result of a LOAD or INCLUDE command. When a file is loaded, the loader checks for unresolved references. If there are any, the loader searches your disks for TEXT files with filenames that match the external entry name. When it finds a match, it loads the TEXT file into storage. If a TEXT file is not found, the loader searches any available TXTLIBs for members that match. If a match is found, it loads the member.

If there are still unresolved references, for example, if you load a program that calls routines PRINT and ANALYZE but the loader cannot locate them, you receive the message:

```
The following names are undefined:
 PRINT
 ANALYZE
```

You can issue the INCLUDE command to load additional TEXT files or TXTLIB members into storage so the loader can resolve any remaining references. For example, if you did not identify the TXTLIB that contains the routines you want to call, you may enter the GLOBAL command followed by the INCLUDE command:

```
global txtlib newlib
include print analyze (start
```

A failure to resolve external references might occur if you have TEXT files with filenames that are different from either the CSECT names or the entry names. You must explicitly issue LOAD and INCLUDE commands for these files.

At execution time, if there are still any unresolved references, their addresses are all set to 0 by the loader; so any attempt to address them in a program may result in a program check.

Controlling the CMS Loader

The LOAD and INCLUDE Commands

The INCLUDE command has the same format and option list (with one exception) as the LOAD command. The main difference is that when you issue the INCLUDE command the loader tables are not reset. If you issue two LOAD commands in succession, the second LOAD command cancels the effect of the first and the pointers to the files loaded are lost.

Conversely, the INCLUDE command, which you must issue when you want to load additional files into storage, should not be used unless you have just issued a LOAD command. You may specify as many INCLUDE commands as necessary following a LOAD command to load files into storage.

The LOAD and INCLUDE commands allow you to specify a number of options. You can:

- Change the entry point to which control is to be passed when execution begins (RESET option).
- Specify the location in virtual storage at which you want the files to be loaded (ORIGIN option).
- Control how CMS resolves references and handles duplicate CSECT names (AUTO, LIBE, and DUP options).
- Clear storage to binary zeros before loading files (CLEAR option). Otherwise, CMS does not clear user storage.
- Save the relocation information from the text files (RLDSAVE option). If the RLDSAVE option is not specified on the LOAD and INCLUDE commands, the relocation information will not be saved for the files being loaded into storage.
- Save history information from the text files (HIST option). If the HIST option is not specified on the LOAD or INCLUDE commands, history information (comments) is not saved for the files being loaded into storage.

When the LOAD and INCLUDE commands execute, they produce a load map, indicating the entry points loaded and their virtual storage locations. You may find this load map useful in debugging your programs. If you do not specify the NOMAP option, the load map is written onto your A-disk in a file named LOAD MAP A5. Each time you issue the LOAD command, the old file LOAD MAP is erased and the new load map replaces it. If you do not want to produce a load map, specify the NOMAP option.

You can find details about these options under the LOAD command in the *VM/SP CMS Command Reference.*

Loader Control Statements

In addition to the options provided with the LOAD and INCLUDE commands that assist you in controlling the execution of TEXT files, you can also use loader control statements. These can be inserted in TEXT files, using the CMS editor. The loader control statements allow you to:

- Set the location counter to specify the address where the next TEXT file is to be loaded (SLC statement).
- Modify instructions and constants in a TEXT file, and change the length of the TEXT file to accommodate modifications (Replace and Include Control Section statements).
- Change the entry point (ENTRY statement).

- Nullify an external reference so that it does not receive control when it is called, and you do not receive an error message when it is encountered (LIBRARY statement).

These statements are also described under the LOAD command in the *VM/SP CMS Command Reference.*

Determining Program Entry Points

When you load a single TEXT file or a TXTLIB member into storage for execution, the default entry point is the first CSECT name in the object file loaded. You can start execution at a different entry point by specifying the entry point on the LOAD (or INCLUDE) command with the RESET option.

```
load myprog (reset beta
```

where BETA is the alternate entry point of your program, or you can specify the entry point on the START command line:

```
start beta
```

When you load multiple TEXT files (either explicitly or implicitly by allowing the loader to resolve external references), you also have the option of specifying the entry point on the LOAD, INCLUDE, or START command lines.

If you do not specifically name an entry point, the loader determines the entry point for you according to the following hierarchy:

1. An entry point specified on the START command
2. The last entry specified with the RESET option on a LOAD or INCLUDE command
3. The name on the last ENTRY statement that was read
4. The name on the last LDT statement that contained an entry name that was read
5. The name on the first assembler- or compiler-produced END statement that was read
6. The first byte of the first control section loaded.

For example, if you load a series of TEXT files that contain no control statements and do not specify an entry point on the LOAD, INCLUDE, or START commands, execution begins with the first file that you loaded. If you want to control the execution of program subroutines, you should be aware of this hierarchy when you load programs or when you place them in TXTLIBs.

An area of particular concern is when you issue a dynamic load (with the OS LINK, LOAD, or XCTL macros) from a program, and you call members of CMS TXTLIBs. The CMS loader determines the entry point of the called program and returns the entry point to your program. If a TXTLIB member that you load has a VCON to another TXTLIB member, the LDT card from the second member may be the last LDT card read by the loader. If this LDT card specifies the name of the second member, CMS may return that entry point address to your program rather than the address of the first member.

Creating Program Modules

When your programs are debugged and tested, you can use the LOAD and INCLUDE commands, in conjunction with the GENMOD command, to create program modules. A module is a file whose external references have been resolved. In CMS, these files must have a filetype of MODULE.

To create a nonrelocatable module file, load the TEXT files or TXTLIB members into storage and issue the GENMOD command:

```
load create analyze print
genmod process
```

The module is generated at the virtual storage address where it is loaded. In this example, PROCESS is the name of the module file, and it has a filetype of MODULE. You could use any name; if you use the name of an existing MODULE file, the old one is replaced.

To execute the program composed of the source files CREATE, ANALYZE, and PRINT, enter:

```
process
```

If PROCESS requires input and/or output files, you have to define these files before PROCESS can execute properly. If PROCESS expects arguments passed to it, you can enter them following the MODULE name. For example,

```
process test1
```

If you want to call your own programs or CMS program modules using SVC 202 instructions, you must be careful not to execute a module that uses the same area of storage that your program occupies. If you want to call a module that executes at location X'20000', you can load the calling program at a higher location. For example,

```
load create (origin 30000
```

As long as the MODULE file called by CREATE is no longer than X'10000' bytes, it will not overlay your program.

You can also use the LOAD and GENMOD commands to create a relocatable CMS module file. However, you must specify the RLDSAVE option in the LOAD command:

```
load progone (RLDSAVE
genmod progtwo
```

The relocatable CMS module file may now be established as a nucleus extension by issuing the NUCXLOAD command:

```
nucxload progtwo
```

Relocatable CMS module files may also be used with the LOADMOD command (for example, when issued as a command from the console). No relocation is performed when the LOADMOD command is used. Relocation is performed only when loaded by NUCXLOAD.

You can use the LOAD, INCLUDE, and GENMOD commands to create a module that includes history information (comments) from the text file used. For example:

```
load progone (HIST
include progtwo (HIST
genmod
```

The generated module contains the comments that were in the text files `progone` and `progtwo`.

Note: Many CMS disk-resident command modules execute in the user program area. That is, if you call a CMS command that runs in the user program area, you must be certain that it does not overlay your own program. Some CMS command modules issue the STRINIT macro or were created using the STR option of the GENMOD command.

Both cause the user area storage pointers to be reset. The reset condition may cause errors upon return to the original program (for example, when OS GETMAIN/FREEMAIN macros are issued in the user program).

The CMS commands that execute in the user program area or that reset the user area storage pointers are identified in the *VM/SP CMS Command Reference.*

The Transient Program Area

To avoid overlaying programs executing in the user program area, you can generate program modules to run in the CMS transient area, which is a two-page area of storage reserved for the execution of programs that are called frequently. Many CMS commands run in this area, which is located at X'E000'. Programs that execute in this area run disabled.

To generate a module to run in the transient area, use the ORIGIN TRANS option when you load the TEXT file into storage, then issue the GENMOD command. For example,

```
load myprog (origin trans
genmod setup (str
```

Note: If a program running in the user area calls a transient routine in which a module was generated using the GENMOD command with the STR option, the user area storage pointers are reset. This reset condition could cause errors upon return to the original program (for example, when OS GETMAIN/FREEMAIN macros are issued in the user program).

The two restrictions placed on command modules executing in the transient area are:

1. They may have a maximum size of 8192 bytes (the size of the transient area).

2. They must be serially reusable. When a program is called by an SVC 202 and if it is already loaded into the transient area, it is not reloaded.

The CMS commands that execute in the transient area are identified in the *VM/SP CMS Command Reference.*

Creating EXEC Procedures

Depending on how you code your programs and EXECs, you can choose whether or not they will be recognized for translation into other languages. CMS only recognizes translations for commands entered from the command line (or with ADDRESS CMS from REXX or &PRESUME &SUBCOMMAND CMS from EXEC 2). CMS does not translate your command name or keywords if you SET TRANSLATE OFF or if you invoke the command from another program using SVC 202 (or with ADDRESS COMMAND from REXX or &PRESUME &COMMAND CMS from EXEC 2). For more information on command translation, refer to "Chapter 7. Developing Commands and Message Files" on page 113.

During your program development and testing cycle, you may want to create EXEC procedures to contain sequences of CMS commands that you execute frequently. For example, if you need a number of MACLIBs, TXTLIBs, and file definitions to execute a particular program, you might have an EXEC procedure as follows:

```
/* EXEC to set up environment to run program TESTA */

signal on error
'GLOBAL MACLIB TESTLIB OSMACRO OSMACRO1'
'ASSEMBLE TESTA'
'PRINT TESTA LISTING'
'GLOBAL TXTLIB TESTLIB PROGLIB'
'ACCESS 200 E'
push 'OS.TEST3.STREAM.BETA'
'FILEDEF INDD1 E DSN ?'
'FILEDEF INDD2 READER'
'FILEDEF OUTFILE DISK TEST DATA A1'
signal off error
'LOAD TESTA (START'
select
  when rc = 100 then do
    .
    .
    .
  end
  when rc = 200 then do
    .
    .
    .
  end
  otherwise
    exit rc
end

Error:
  say 'Error occurred on line' sigl':' sourceline(sigl)
  exit rc
```

The "signal on error" control statement in the EXEC procedure ensures that if an error occurs during any part of the EXEC, the remainder of the EXEC does not execute, and the "Error:" displays the line number where the error occurred as well as the actual command which gave the error.

Note: For the FILEDEF command entered with the DSN ? operand, you must stack the response (using "push") before issuing the FILEDEF command.

When your program is finished executing, the REXX special variable RC indicates the contents of general register 15 at the time the program exited (the "Return Code"). You can use this value to perform additional steps in your EXEC procedure. Additional steps are indicated in the preceding example by ellipses.

CMS Macro Instructions

There are a number of assembler language macros distributed with the CMS system that you can use when you are writing programs to execute in the CMS environment. These macros are in the macro libraries CMSLIB MACLIB and DMSSP MACLIB, which are normally located on the system disk.

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

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

There are macros to manipulate CMS disk files, to handle terminal communications, to manipulate unit record and tape input/output, and to trap interruptions. These macros are discussed in general terms here. For complete format descriptions, see the *VM/SP CMS Macros and Functions Reference.*

Disk File Manipulation

Disk files are described in CMS by means of a file system control block (FSCB). The CMS macro instructions that manipulate disk files use FSCBs to identify and describe the files. When you want to manipulate a CMS file, you can refer to the file by its file identifier, specifying 'filename filetype filemode' in quotation marks, or you can refer to the FSCB for the file, specifying FSCB=fscb, where fscb is the label on an FSCB macro.

To establish an FSCB for a file, use the FSCB macro instruction specifying a file identifier. For example,

```
INFILE   FSCB  'INPUT TEST A1'
```

You can also provide, on the FSCB macro instruction, descriptive information to be used by the input and output macros. If you do not code an FSCB macro instruction for a file, an FSCB is created in-line (following the macro instruction) when you code an FSREAD, FSWRITE, or FSOPEN macro instruction.

The format of an FSCB and a description of each of the fields are listed below.

Label	Description
`FSCBCOMM DC      CL8' '`	File system command

Figure 10 (Part 1 of 2). FSCB Format

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

Figure 10 (Part 2 of 2). FSCB Format

The FSCBAITN, FSCBANIT, FSCBWPTR, and FSCBRPTR fields are only generated in the FSCB when the extended format FSCB is requested (FORM=E is coded on the FSCB macro instruction). In this case, the FSCBITNO and FSCBNOIT fields are reserved fields. Extended format FSCBs must be used to manipulate files larger than 65,533 items. The labels shown above are not generated by the FSCB macro. To reference fields within the FSCB by these labels, you must use the FSCBD macro instruction to generate a DSECT.

FSCBCOMM: When the FSCBFN, FSCBFT, and FSCBFM fields are filled in, you can fill in the FSCBCOMM field with the name of a CMS command and use the FSCB as a parameter list for an SVC 202 instruction. (You must place a delimiter to mark the end of the command line.)

FSCBFN, FSCBFT, FSCBFM: The filename, filetype, and filemode fields identify the CMS file to be read or written. You can code the fileid on a macro line in the format 'filename filetype filemode', or you can use register notation. If you use register notation, the register you specify must point to an 18-byte field in the format:

```
FILEID   DC    CL8'filename'
         DC    CL8'filetype'
         DC    CL2'filemode'
```

The fileid must be specified either in the FSCB for a file or on the FSREAD, FSWRITE, FSOPEN, or FSERASE macro instruction that references the file.

FSCBITNO: For an FSCB without the FORM = E option, the record or item number indicates the relative record number of the next record to be read or written. It can be changed with the RECNO option. The default value for this field is 0. When you are reading a file, a 0 indicates that records are to be read sequentially beginning with the first record in the file. When you are writing a file and the file already exist, a 0 indicates that records are to be written sequentially beginning at the first record following the end of the file. If the file is a new file, begin writing at record 1.

For an FSCB generated with the FORM = E option, the FSCBAITN field contains the record or item number. The FSCBITNO field is reserved.

Whenever you read discontiguous files in CMS (that is, files with missing records), the input buffer will be filled with the appropriate number of bytes. Be aware that the flag byte in the FSCB may not reflect whether the input buffer contains generated data items from RDBUF.

FSCBBUFF: The buffer address, specified in the BUFFER option, indicates the label of the buffer where the record is to be written from or where the record is to be read into. You should always supply a buffer large enough to accommodate the longest record you expect to read or write. This field must be specified either in the FSCB or on the FSREAD or FSWRITE macro instruction.

FSCBSIZE: This field indicates the number of bytes that are read or written with each read or write operation. The default value is 0. If the buffer that you use represents the full length of the records you are going to be reading or writing, you can use the BSIZE option to set this field equal to your buffer length. When you are writing variable-length records, use the BSIZE operand to indicate the length of each record you write. This field must be specified.

FSCBFV: This two-character field indicates the record format (RECFM) of the file. The default value is F (fixed).

FSCBFLG: The flag byte is X'20' indicating an extended FSCB is generated when the FORM = E option is coded on the FSCB macro instruction.

FSCBNOIT: For an FSCB without the FORM = E option, this field contains the number of whole records to be read or written in each read or write operation. You can use the NOREC option with the BSIZE option to block and deblock records.

For an FSCB generated with the FORM = E option, the FSCBANIT field contains the number of whole records to be read or written. The FSCBNOIT field is reserved.

FSCBNORD: Following a read operation, this field contains the number of bytes actually read, so if you are reading a variable-length file, you can determine the size of the last record read. The FSREAD macro instruction places the information from this field into register 0.

FSCBAITN: The alternate record or item number indicates the relative record number of the next record to be read or written in an extended FSCB format. See the description of the FSCBITNO field for the usage of this field.

FSCBANIT: This field contains the alternate number of whole records in an extended FSCB format. See the description of the FSCBNOIT field for the usage of this field.

FSCBWPTR: The FSPOINT macro instruction uses this field to contain the alternate write pointer for an extended FSCB during a POINT operation.

FSCBRPTR: The FSPOINT macro instruction uses this field to contain the alternate read pointer for an extended FSCB during a POINT operation.

Using the File System Control Block (FSCB)

The following example shows you how to code an FSCB macro instruction to define various file and buffer characteristics and how to use the same FSCB to refer to different files:

```
         FSREAD 'INPUT FILE A1',FSCB=COMMON,FORM=E
         FSWRITE 'OUTPUT FILE A1',FSCB=COMMON,FORM=E
         .
         .
         .
COMMON   FSCB   BUFFER=SHARE,RECFM=V,BSIZE=200,FORM=E
SHARE    DS     CL200
```

In the above example, the fileid specifications on the FSREAD and FSWRITE macro instructions modify the FSCB at the label COMMON each time a read or write operation is performed. You can also modify an FSCB directly by referring to fields by a displacement off the beginning of the FSCB. For example,

```
MVC    FSCB+8,=CL8'NEWNAME'
```

moves the name NEWNAME into the filename field of the FSCB at the label FSCBFN.

As an alternative, you can use the FSCBD macro instruction to generate a DSECT and refer to the labels in the DSECT to modify the FSCB. For example,

```
         LA     R5,INFSCB
         USING  FSCBD,R5
         .
         .
         MVC    FSCBFN,NEWNAME
         .
         .
INFSCB   FSCB   'INPUT TEST A1',FORM=E
NEWNAME  DC     CL8'OUTPUT'
         FSCBD
```

In the above example, the MVC instruction places the filename OUTPUT into the FSCBFN (filename) field of the FSCB. The next time this FSCB is referenced, the file OUTPUT TEST is the file that is manipulated.

Reading and Writing CMS Disk Files

CMS disk files are sequential files. When you use CMS macros to read and write these files, you can access them sequentially with the FSREAD and FSWRITE macros. However, you may also refer to records in a CMS file by their relative record numbers. So you can, in effect, access records using a direct access method.

If you know the record you want to read or write, you can specify the RECNO option on the FSCB macro instruction or on the FSOPEN, FSREAD, or FSWRITE macro instructions. When you use the RECNO option on the FSCB macro instruction, you must specify the exact record number. For the FSOPEN, FSREAD, or FSWRITE macro instructions, you may either specify the exact record number:

```
WRITE    FSWRITE FSCB=WFSCB,RECNO=10,FORM=E
```

or specify the register containing the record to be read:

```
WRITE    FSWRITE FSCB=WFSCB,RECNO=(5),FORM=E
```

When you want to access files sequentially, the FSCBITNO field of the FSCB must be 0 for an FSCB without the FORM=E option. For an extended FSCB, the FSCBAITN field must be 0. This is the default value. When you are reading files with the FSREAD macro instruction, reading begins with record number 1. When you are writing records to an existing file with the FSWRITE macro, writing begins following the last record in the file.

To begin reading or writing files sequentially beginning at a specific record number, you must specify the RECNO option twice: once to specify the relative record number where you want to begin reading, and a second time to specify RECNO=0 so reading or writing will continue sequentially beginning after the record just read or written. You can specify the RECNO option on the FSREAD or FSWRITE macro instruction, or you may change the FSCBITNO or FSCBAITN field in the FSCB for the file, as necessary for the FSCB form.

For example, to read the first record and then the 50th record of a file, you could code the following:

```
READ1    FSREAD FSCB=RFSCB,FORM=E
         FSWRITE FSCB=WFSCB,FORM=E
         LA    5,RFSCB
         USING FSCBD,5
         MVC   FSCBAITN,=F'50'
READ50   FSREAD FSCB=RFSCB,FORM=E
         FSWRITE FSCB=WFSCB,FORM=E
         .
         .
         .
RFSCB    FSCB 'INPUT FILE A1',BUFFER=COMMON,BSIZE=120,FORM=E
WFSCB    FSCB 'OUTPUT FILE A1',BUFFER=COMMON,BSIZE=120,FORM=E
COMMON   DS    CL120
         .
         .
         .
         FSCBD
```

In this example, the statements at the label READ1 write record 1 from the file INPUT FILE A1 to the file OUTPUT FILE A1. Then, using the DSECT generated by the FSCBD macro, the FSCBITNO field is changed because an extended FSCB is being used.

FSCBAITN field is changed because an extended FSCB is being used and record 50 is read from the input file and written into the output file.

The "update-in-place" facility allows you to write blocks back to their previous location on disk. The "update-in-place" attribute of a CMS file is indicated by the filemode number 6.

Reading and Writing Variable Length Records

When you read or write variable-length records, you must specify RECFM=V either in the FSCB for the file or on the FSWRITE or FSREAD macro instruction. The read/write buffer should be large enough to accommodate the largest record you are going to read or write.

To write variable-length records, use the BSIZE= option on the FSWRITE macro instruction to indicate the record length for each record you write. To read variable-length records, register 0 contains, on return from FSREAD, the length of the record read.

The following example shows how you could read and write a variable-length file:

```
READ   FSREAD 'DATA CHECK A1',BUFFER=SHARE,BSIZE=130,
         ERROR=OUT,FORM=E
       FSWRITE 'COPY DATA A1',BUFFER=SHARE,BSIZE=(0),FORM=E
       B     READ
```

When you update files of variable-length records, the replacement record must be the same length as the original record. An attempt to write a record shorter or longer than the original record results in truncation of the file at the specified record number. No error return code is given.

End-of-File Checking

You can specify the ERROR= operand with the FSREAD or FSWRITE macro instruction so an error handling routine receives control in case of an error. In CMS, when an end of file occurs during a read request, it is treated as an error condition. The return code is always 12. If you specify an error handling routine on the FSREAD macro instruction, the first thing this routine can do is check for a 12 in register 15.

Your error handling routine may also check for other types of errors. See the macro description in the *VM/SP CMS Macros and Functions Reference* for details on the possible errors and the associated return codes.

Opening and Closing Files

Usually, CMS opens a file whenever an FSREAD or FSWRITE macro instruction is issued for the file. When control returns to CMS from a calling program, all files accidentally left open are closed by CMS. Therefore, you do not have to close files at the end of a program.

For a minidisk in 512-, 1K-, 2K-, or 4K-byte block format, a file may be open for concurrent read and write operations and an FSCLOSE need not be issued when switching from reading to writing, or vice versa. For example:

```
        LA     3,2

READ    FSREAD FSCB=UPDATE,RECNO=(3),ERROR=READERR,FORM=E
        .
        .
        .
        FSWRITE FSCB=UPDATE,RECNO=(3),ERROR=WRITERR,FORM=E
        LA     3,1(3)
        B      READ
        .
        .
        .
UPDATE  FSCB   'UPDATE FILE A1',BUFFER=BUF1,BSIZE=80,FORM=E
```

When you are running long running applications or running disconnected, include several FSCLOSE macros to each file referenced. This insures that changes to the file are reflected on the disk in the event that the user is forced off the system. This consideration is important when running on 512-, 1K-, 2K-, or 4K-byte block disks since the disk directory is not updated until all of the files on the disk are closed.

If you want to read and write records from the same file on an 800-byte block format minidisk, you must issue an FSCLOSE macro instruction to close the file whenever you switch from reading to writing. For example:

```
        LA     3,2
READ    FSREAD FSCB=UPDATE,RECNO=(3),ERROR=READERR
        FSCLOSE FSCB=UPDATE
        .
        .
        .
        FSWRITE FSCB=UPDATE,RECNO=(3),ERROR=WRITERR
        FSCLOSE FSCB=UPDATE
        LA     3,1(3)
        B      READ
        .
        .
        .
UPDATE  FSCB   'UPDATE FILE A1',BUFFER=BUF1,BSIZE=80
```

To execute a loop to read, update, and rewrite records, you must read a record, close the file, write a record, close the file, and so on. Since closing a file repositions the read pointer to the beginning of the file and the write pointer at the end of the file, you must specify the relative record number (RECNO) for each read and write operation. In the above example, register 3 is used to contain the relative record number. It is initialized to begin reading with the second record in the file and is increased by one following each write operation.

When you use an EXEC to execute a program to read or write a file, the file is not closed by CMS until the EXEC completes execution. Therefore, if you read or write the same file more than once during the EXEC procedure, you must use an FSCLOSE macro instruction to close the file after using it in each program, or you must use the FSOPEN macro instruction to open it before each use. Otherwise, the read or write pointer is positioned as it was when the previous program completed execution.

Creating New Files

When you want to begin writing a new file using CMS data management macros, there are two ways to ensure that the file you want to create does not already exist. One way is to issue the FSSTATE macro instruction to verify the existence of the file.

A second way to ensure that a file does not already exist is to issue an FSERASE macro instruction to erase the file. If the file does not exist, register 15 returns with a code of 28. If the file does exist, it is erased. See Figure 11 on page 83 for an illustration of a sample program using CMS data management macros.

```
LINE    SOURCE STATEMENT

BEGIN    CSECT      1
         PRINT NOGEN
         USING *,12          ESTABLISH ADDRESSABILITY
         LR    12,15
         ST    14,SAVE
         LA    2,8(,1) R2=ADDR OF INPUT FILEID IN PLIST     2
         LA    3,32(,1) R3=ADDR OF OUTPUT FILEID IN PLIST
* DETERMINE IF INPUT FILE EXISTS
         FSSTATE (2),ERROR=ERR1,FORM=E
*
* READ A RECORD FROM INPUT FILE AND WRITE ON OUTPUT FILE
RD       FSREAD (2),ERROR=EOF,BUFFER=BUFF1,BSIZE=80,FORM=E     3
         FSWRITE (3),ERROR=ERR2,BUFFER=BUFF1,BSIZE=80,FORM=E
         B     RD            LOOP BACK FOR NEXT RECORD
*
* COME HERE IF ERROR READING INPUT FILE
EOF      C     15,=F'12'     END OF FILE ?     4
         BNE   ERR3          ERROR IF NOT
         LA    15,0          ALL O.K. - ZERO OUT R15
         B     EXIT          GO EXIT
* IF INPUT FILE DOES NOT EXIST
ERR1     WRTERM 'FILE NOT FOUND',EDIT=YES
         B     EXIT
*
* IF ERROR WRITING FILE
ERR2     LR    10,15         SAVE RET CODE IN REG 10     5
         LINEDIT TEXT='ERROR CODE .... IN WRITING FILE',SUB=(DEC,(10))
         B     EXIT
*
* IF READING ERROR WAS NOT NORMAL END OF FILE
ERR3     LR    10,15         SAVE RET CODE IN REG 10     5
         LINEDIT TEXT='ERROR CODE .... IN READING FILE',SUB=(DEC,(10))
*
EXIT     L     14,SAVE       LOAD RETURN ADDRESS
         BR    14            RETURN TO CALLER
*
BUFF1    DS    CL80
SAVE     DS    F
         END
```

Figure 11 (Part 1 of 2). A Sample Listing of a Program that Uses CMS Macros

Notes:

1 The program might be invoked with a parameter list in the format: progname INPUT FILE A1 OUTPUT FILE A1. This line is placed in a parameter list by CMS routines and addressed by register 1 (see note 2).

2 The parameter list is a series of doublewords, each containing one of the words entered on the command line. Thus, 8 bytes past register 1 is the beginning of the input fileid. 24 bytes beyond that is the beginning of the second fileid.

3 The FSREAD and FSWRITE macros cause the files to be opened. No open macro is necessary. CMS routines close all open files when a program completes execution (except CMS EXEC files).

4 The return code in register 15 is tested for the value 12, indicating an end-of-file condition. If it is the end of the file, the program exits. Otherwise, it writes an error message.

5 The dots in the LINEDIT macro are substituted, during execution, with the decimal value in register 10.

Figure 11 (Part 2 of 2). A Sample Listing of a Program that Uses CMS Macros

Terminal Communications

There are four CMS macros you can use to write interactive, terminal-oriented programs. They are RDTERM, WRTERM, LINEDIT, and WAITT. RDTERM and WRTERM only require a read/write buffer for sending and receiving lines from the terminal. The third, LINEDIT, has a substitution and translation capability.

When you use the WRTERM macro to write a line to your terminal you can specify the actual text line in the macro instruction, for example:

```
DISPLAY  WRTERM 'GOOD MORNING'
```

You can also specify the message text by referring to a buffer that contains the message.

The RDTERM macro accepts a line from the terminal and reads it into a buffer you specify. You could use the RDTERM and WRTERM macros together, as follows:

```
WRITE      WRTERM 'ENTER LINE'
READ       RDTERM BUFFER
           LR   3,0
REWRITE    WRTERM BUFFER,(3)
             .
             .
             .
BUFFER     DS   CL130
```

In this example, the WRTERM macro results in a prompting message. Then the RDTERM macro accepts a line from the terminal and places it in the buffer BUFFER. The length of the line read, contained in register 0 on return from the RDTERM macro, is saved in register 3. When you specify a buffer address on the WRTERM macro instruction, you must specify the length of the line to be written. Here, register notation is used to indicate that the length is contained in register 3.

The LINEDIT macro converts decimal and hexadecimal data into EBCDIC, and places the converted value into a specified field in an output line. There are list and execute forms of the macro instruction, which you can use in writing reentrant code. Another option allows you to write lines to the offline printer. The LINEDIT macro is described, with examples, in *VM/SP CMS Macros and Functions Reference.* Figure 11 on page 83 shows how you might use the LINEDIT macro to convert and display CMS return codes.

The WAITT (wait terminal) macro instruction can help you to synchronize input and output to the terminal. If you are executing a program that reads and writes to the terminal frequently, you may want to issue a WAITT macro instruction to halt execution of the program until all terminal I/O has completed.

Unit Record and Tape I/O

CMS provides macros to simplify reading and punching cards (RDCARD and PUNCHC), and creating printer files (PRINTL). When you use either the PUNCHC or PRINTL macros to write or punch output files while a program is executing, you should remember to issue a CLOSE command for your virtual printer or punch when you are finished. You can do this either after your program returns control to CMS, by entering:

```
cp close e

-- or --

cp close d
```

or you can set up a parameter list with the command line CP CLOSE E or CP CLOSE D and issue an SVC 202.

The tape control macros, RDTAPE, WRTAPE and TAPECTL, can read and write CMS files from tape, or control the positioning of a tape.

Handling Interrupts

You can set up routines in your programs to handle interruptions caused by I/O devices, by SVCs, or by external interruptions using the HNDINT, HNDSVC, or HNDEXT macro instructions.

With the HNDINT macro instruction, you can specify addresses that are to receive control when an interruption occurs for a specified device. If the

WAIT option is used for a device specified in the HNDINT macro instruction, then the interruption handling routine specified for the device does not receive control until after the WAITD macro instruction is issued for the device.

You can use the HNDSVC macro instruction to trap supervisor call instructions of particular numbers, if, for example, you want to perform some additional function before passing control or you do not want any SVCs of the specified number to be executed.

The CP EXTERNAL command simulates external interruptions in your virtual machine; if you want to be able to pass control to a particular internal routine in the event of an external interruption, you can use the HNDEXT macro instruction.

System Product Editor Interface to Access Files in Storage

CMS uses the SUBCOM facility to allow a number of CMS commands to use an XEDIT interface to access files in storage. Applications can read or write specific records without having to go to disk or use the program stack to transfer the data to or from XEDIT. This improves performance.

CMS uses the XEDIT interface for processing the FILELIST, HELP, MACLIST, PEEK, and SENDFILE commands. The interface is invoked by specifying the XEDIT option on the LISTFILE, MACLIB, or NAMEFIND commands. This option may only be specified from the XEDIT environment.

When using this interface from an application program, only the extended parameter list can be used, and the high-order byte of of register 1 must contain X'02' to indicate SUBCOM is being used.

The application can invoke this interface via SVC 202 or via a BALR instruction. Because XEDIT is a nucleus-resident routine, other nucleus-resident routines can branch directly to it while routines that do not reside in the nucleus use SVC linkage. When using an SVC 202, register 1 must point to the FSCB where the name of the routine being invoked is the first token. The high-order byte of register 1 must also be X'02'. When using BALR, the calling program can determine the entry point it wants by using SUBCOM. In this case, register 1 points to the FSCB and register 2 points to the SCBLOCK. The address of the the SCBLOCK has been returned from SUBCOM.

The routines available, their entry point names, and error return codes are:

- DMSXFLST - This routine returns the characteristics of a file (RECFM, LRECL, etc). It also ensures that the file is in the XEDIT ring. The return codes are:

0	File is in the XEDIT ring
24	Incomplete fileid specified

28	File is not in the XEDIT ring

Note: Return codes are similar to those for ESTATE.

- DMSXFLRD - This routine transfers one record from XEDIT storage to the calling program. If RECFM=F, it may transfer more than one record. The return codes are:

0	READ performed
1	File is not in the XEDIT ring
2	Invalid buffer address
5	Number of items equals zero
7	RECFM is not 'F' or 'V'
8	Buffer is too small (records truncated)
11	Number of items is not equal to one for V-file
12	End of file

Note: Return codes are similar to those for FSREAD.

- DMSXFLWR - This routine transfers one record from the calling program to XEDIT storage. If RECFM=F, it may transfer more than one record. The return codes are:

0	WRITE performed
2	User buffer address equals zero
7	Skip over unwritten records
8	Number of bytes is not specified
11	RECFM is not 'F' or 'V'
13	No more space is available
14	Number of bytes is not integrally divisible by the number of item
15	Item length is not the same as previous
16	RECFM of 'F' or 'V' is not the same as previous
18	Number of items is not equal to one for V-file
28	File is not in the XEDIT ring

Note: Return codes are similar to those for FSWRITE.

- DMSXFLPT - This routine moves the current line pointer to a record specified by the calling program. If you specify the read and write pointer as all ones (X'FFFFFFFFX'), the current line pointer is returned in the FSCB. The return codes are:

0	POINT performed
1	File not found
2	Invalid FSCB

Note: Return codes are similar to those for FSPOINT.

When the interface is used, XEDIT determines if a file is in the XEDIT ring (active in storage) and does the processing required. The files in the XEDIT ring are always open. New files may be added to the ring with the XEDIT subcommand. Files in the ring may be closed with the FILE or QUIT subcommands.

The current line pointer serves the function of both the read and write pointers of the CMS file system. If RECNO=0 is specified in a call to DMSXFLRD, the data is transferred to the calling program starting at the current line pointer. Transfer is stopped when the specified number of lines has been transferred or when end-of-file is reached. The current line pointer is advanced by one for each record transferred to the calling program. If the current line pointer was at the end-of-file when DMSXFLRD was called, no data is transferred and an end-of-file condition is returned.

If RECNO=0 is specified in a call to DMSXFLWR, new records are written starting at the line pointed to by the current line pointer. These new records replace any existing records or add new records if at the end-of-file. The current line pointer is advanced to the line following the last line written at the end of the operation. Note that writing to a record in the middle of a V-format file does not result in truncation of the file from that point as it would in the CMS file system. Truncation (or spilling when SET SPILL ON|WORD) may occur if the file is in V-format and the LRECL of the file is less than the length of the record(s) being written. No message is issued and the return code is 0.

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.

The console facility provides a CMS macro interface to full-screen I/O that:

- provides screen coordination and
- provides an architecturally independent I/O interface.

Use the console facility instead of the DIAGNOSE code X'58' interface or the DISPW macro.

The console facility provides improved usability for writing 3270 I/O applications. The CONSOLE macro performs I/O operations such as:

- building the channel command word (CCW),
- issuing DIAGNOSE code X'58' or Start I/O (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, full-screen 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. Refer to "The CONSOLE Macro" on page 90 for more details.

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

The four formats of the CONSOLE macro instruction are:

Standard format
: It is not reentrant.

List format (MF=L)
: Generates a parameter list, but does not generate code to execute the function. The parameter list is generated in-line and usually register notation cannot be used.

Complex List format (MF=(L,addr[,label]))
Generates a parameter list, but does not generated code to execute the function. The parameter list is generated in an area that you specify.

Execute format (MF=(E,addr))
Generates code to execute the function.

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

The CONSOLE Macro

An Example of Using the Console Facility

- OPEN a path with the optional BUFFER parameter.
- Get information about the device from the buffer.
- Build a 3270 data stream.
- Issue the CONSOLE WRITE with the EW option.
- Issue the READ with the WAIT option.
- Check the return code.
- If the return code from READ or WRITE is not 0, issue a QUERY to determine what happened.
- CLOSE the path.

Opening a Path

In order to use the CMS console facility for I/O, you must first open a 'path'. You can do this by issuing the OPEN parameter of the CONSOLE macro.

When you open a path, the console facility allocates storage containing information about I/O activity for the application. A path entry is associated with a device when you specify the CONSOLE OPEN parameter. If the device does not already have paths opened to it, storage is allocated for a new device entry and any existing device entries are linked to it.

Querying a Path

You can use the CONSOLE QUERY to obtain information about a path and its associated device or to obtain information about a virtual device even if paths are not opened to it. To do this, you must also specify the BUFFER parameter. You can map the information returned by the CQYSECT macro. (See *VM/SP Data Areas and Control Block Logic Volume 2 (CMS)* for more information.) CQYSECT contains length equates that an application can use to obtain the size buffer needed.

Initially, when an application opens a path, it can specify a buffer that contains path and device information and is mapped by CQYSECT. This information is very useful at initialization time since it contains DIAGNOSE code X'24' information, and, depending on the device, it contains DIAGNOSE code X'8C' information. Then, the application can obtain the device type and characteristics and can use the appropriate routines to build data streams.

Writing to and Reading from a Path

The console facility keeps track of the application that owns the screen by keeping a field in the device entry (CDEV) for the address of the path entry that performed the last I/O operation. If one application currently owns the screen and another application wants to perform I/O, the second application must gain control of the screen by reformatting it with an erase/write (EW), an erase/write alternate (EWA), or, in some cases, a write structure field (WSF). Therefore, applications should begin I/O processing with one of these operations.

> ***Warning:*** Until all applications use the console facility instead of issuing a DIAGNOSE code X'58', there is a possibility of seeing data from two different applications mixed on the screen. An erase/write (EW) must be issued so that the current application can gain total ownership of the screen. If CP breaks in and writes a screen or if another application using the console facility writes a screen, the console facility can detect this situation and issues return code 32. If you get return code 32, issue an erase/write or an erase/write alternate. However, the console facility can not always detect who wrote to the screen when applications modify PSWs in low storage and issue their own DIAGNOSE X'58'. In this case, if you get mixed data on the screen, you will have to press the CLEAR key or issue a command that causes an erase/write. The VMFCLEAR command can be issued by an application before exiting their program to accomplish this. Another alternative for applications running in full-screen CMS would be to write an EXEC to issue a SET FULLSCREEN SUSPEND command, then invoke their full-screen program, and when processing completes they can resume fullscreen CMS by issuing SET FULLSCREEN RESUME.

Most full-screen applications will wait for input and then read the data. To accomplish this, you can issue a CONSOLE READ with the WAIT option, or you can issue a CONSOLE WAIT followed by a CONSOLE READ. The

CONSOLE READ with the WAIT option has a performance advantage because it issues only one Supervisor Call (SVC) instead of two.

If you receive a return code 1 on an I/O operation, you should issue an explicit WAIT. Do not specify a READ with the WAIT option. A disconnect is detected before any READ options are checked. When you specify a READ with the WAIT option, a WAIT is *not* issued and control is immediately returned to the application with return code 1.

If an application performs linemode I/O or calls routines that perform linemode I/O, it should issue a CMS WAITT macro to coordinate linemode and full-screen I/O. This allows the I/O to complete before issuing any console full-screen operations.

I/O Advantages

Applications can become much less dependent on low-level device architecture by using the console facility. In the past, applications not only had to construct data streams, but they had to build the channel command word (CCW), determine the type of device (dialed or virtual console) and set up for DIAGNOSE code X'58' or 3270 SIO, and check the channel status word (CSW) to determine what action should be taken.

In addition, the applications would have to change whenever new devices were introduced.

With the console facility, the application only needs to:

- build a data stream,
- issue a CONSOLE call, and
- check a return code.

The CSW error checking and retrying of I/O is much more elaborate in the console facility than what many applications do today. This gives applications a better chance for completing a successful I/O options.

Building Your Own CCW

The CONSOLE EXCP parameter allows applications that still want to build their own CCWs to do so. However, no CCW error checking is performed so the CONSOLE module cannot determine the type of I/O requested and cannot coordinate the use of the screen as effectively as with the READ or WRITE functions.

The EXCP parameter is not recommended for the virtual console since the application has to know the internal I/O processing performed in the CONSOLE module. However, if you use this function carefully, you can chain several CCWs. The first CCW in the string should be an EW, EWA, or WSF to reformat and to gain control of the screen.

Completing an I/O Operation

When the console facility returns control to the application after an I/O operation, the application can check the return code and continue processing. If more information is needed about the I/O just performed, a CONSOLE QUERY can be issued. The CONSOLE QUERY shows the CSW after I/O, the sense data (if any), the last CCW executed, and all the device information obtained by DIAGNOSE code X'24' and DIAGNOSE code X'8C'.

Closing a Path

Before exiting your program, paths that are no longer needed should be closed. Close processing releases storage for the path entry. If this was the only path opened to the associated device, the device entry storage is released as well. When releasing a device entry, a CP RESET command is only issued for dialed devices.

The DISPW Macro

Although the CONSOLE macro is the preferred interface for full-screen 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* $\left[\text{,LINE}=\left\{\begin{matrix}n\\\underline{0}\end{matrix}\right\}\right]\left[\text{,BYTES}=\left\{\begin{matrix}nnnn\\\underline{1760}\end{matrix}\right\}\right]$ **[,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= $\left\{\begin{matrix}n\\\underline{0}\end{matrix}\right\}$
: 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= $\left\{\begin{array}{l} nnnn \\ \underline{\mathbf{1760}} \end{array}\right\}$

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 data 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 6. Updating Source Programs Using CMS

As you test and modify programs, you may want to keep backup copies of the source programs. Then you can always return to a certain level of a program in case you have an error. CMS provides several approaches to the problem of program backup. The method you choose depends on the complexity of your project, the changes you want to make, and the size of your programs.

The simplest method is to make a copy of the current source file under a new name. You can do this using either the COPYFILE command or the editor.

The UPDATE Philosophy

While the procedures outlined above for modifying programs are suitable for many applications, they may not be adequate in a situation where several programmers are applying changes to the same source code. These procedures also have the drawback of not providing you with a record of what has been changed. After using the editor, you do not have a record of the lines that have been deleted, added, replaced, and so on, unless you manually add comments to the code, insert special characters in the serialization column, or use some technique that records program activity.

The UPDATE command and the XEDIT UPDATE option provide a way for you to modify a source program without affecting the original. UPDATE produces an update log, indicating the changes that have been made. Both UPDATE and XEDIT have the capability of combining multiple updates at one time, so that changes made by different programmers or changes made at different times can be combined into a single output file.

The UPDATE command and the XEDIT UPDATE option are the basic elements of the VM/SP updating scheme. Although the input filetypes used by the UPDATE command default to ASSEMBLE file characteristics, neither the UPDATE command nor the XEDIT UPDATE option is limited to assembler language programs. Also, is it not limited to system programming applications. You can use it to modify and update any fixed-length, 80-character file that does not have data in columns 73 through 80.

Update Files

A simple update involves two input files:

- The source file, which is the program you want to update
- An update file, usually created by XEDIT, containing control statements describing the changes you want to make.

The control statement file usually has a filetype of UPDATE. For convenience, you can give it the same filename as your source file. For example, if you want to update the file SAMPLE ASSEMBLE, you would create a file named SAMPLE UPDATE using the XEDIT UPDATE option. To apply the changes in the update file, you issue the command:

```
update sample
```

The default values used by the UPDATE command are filetypes of ASSEMBLE and UPDATE for the source and update files, respectively. If you are updating a COBOL source program named READY COBOL with an update file named READY UPDATE, you would issue the command:

```
update ready cobol a ready update a
```

After an UPDATE command completes processing, the input files are not changed; two new files are created. One of them contains the updated source file, with a filename that is the same as the original source file but preceded by a dollar sign ($). Another file, containing a record of updates is also created; it has a filename that is the same as the source file and a filetype of UPDLOG. For example:

Source Files	Output Files
SAMPLE ASSEMBLE	$SAMPLE ASSEMBLE
SAMPLE UPDATE	SAMPLE UPDLOG
READY COBOL	$READY COBOL
READY UPDATE	READY UPDLOG

Now, you can assemble or compile the new source file created by the UPDATE command.

Creating an Update File

You can create an update file using the XEDIT UPDATE option. Using XEDIT, you do not need to enter the control statements in the UPDATE file. They are generated automatically by the editor. For example:

```
xedit ready cobol a (upd
```

specifies that a file called READY COBOL is to be edited and all updates to the file are placed in a separate file called READY UPDATE along with the appropriate control statements.

The XEDIT UPDATE option expects source files to have sequence numbers in columns 73 through 80. Before you can create an UPDATE file you must use the XEDIT SERIAL subcommand to sequence your files. To generate these sequence numbers, you should issue:

```
serial all
```

prior to issuing a FILE or SAVE subcommand when you are editing a file. Alternately, you can preface sequence numbers with a three character identifier, usually the first three characters of the filename. If you issue:

```
serial on
```

XEDIT writes sequence numbers in columns 76 through 80 of your file. Columns 73 through 75 contain the first three characters of the filename. If SERIAL ON is specified, you must also specify the NOSEQ8 option on the XEDIT command to tell the editor to expect a sequence of numbers only in columns 75 through 80. For example:

```
xedit ready cobol a (upd noseq8
```

Using an Existing Update File

If an update file already exists for a given source file and you wish to either:

1. browse the source file with the updates applied or

2. continue updating the source file

issue the same XEDIT command that you entered when you created the update file. For example:

```
xedit ready cobol a (upd
```

applies all updates contained in READY UPDATE to the source file READY COBOL and displays the resulting file on the screen. Any updates created during this editing session are added to those already contained in READY UPDATE. Again, all control statements are automatically generated by XEDIT. More information about the XEDIT UPDATE option can be found in the *VM/SP CMS Command Reference.*

UPDATE Control Statements

The control statements used by the UPDATE command are similar to those used by the OS IEBUPDTE utility program or the DOS MAINT program UPDATE function.

Each UPDATE statement must have the characters ./ in columns one and two, followed by one or more blanks. The statements are described below.

SEQUENCE Statement: This statement tells the UPDATE command you want to number or renumber the records in a file. Sequence numbers are written in columns 73 through 80. For example, the statement:

```
 ./ S  1000
```

indicates that you want sequence numbering to be done in increments of 1000 with the first statement numbered 1000. The SEQUENCE statement is convenient if you want to apply updates to a file that does not already have sequence numbers. In this case, you may want to use the REP (replace) option of the UPDATE command, so that instead of creating a new file ($filename), the original source file is replaced:

```
update sample (rep
```

INSERT Statement: This statement precedes new records that you want to add to a source file. The INSERT statement tells the UPDATE command where to add the new records. For example, the lines:

```
 ./ I  1600
 TEST2  TM   HOLIDAY,X'02'    HOLIDAY?
        BNO  VACATION         NOPE...VACATION
```

insert two lines of code, following the statement numbered 00001600, into the output file. The inserted lines are flagged with asterisks in columns 73 through 80. The INSERT statement also allows you to request that new statements be sequenced. See "Sequencing Output Records" on page 99.

DELETE Statement: This statement tells the UPDATE command which records to delete from the source file. If your update file contains:

```
 ./ D  2500
```

then only the record 00002500 is deleted. If the file contains

```
 ./ D  2500  2800
```

then all the statements from 2500 through 2800 are deleted from the source file.

REPLACE Statement: The REPLACE statement replaces one or more records in the source file. It precedes the new records you want to add. It is a combination of the DELETE and INSERT statements. For example, the lines

```
 ./ R 38000 38500
 PLIST    DS    0D
          DC    CL8'TYPE'
          DC    CL8' '
          DC    CL8'FILE'
          DC    CL8'A1'
          DC    8X'FF'
```

replace the existing statements numbered 38000 through 38500 with the new lines of code. As with the INSERT statement, new lines are not automatically resequenced.

COMMENT Statement: Use this statement when you want to place comments in the update log file. For example, the line:

```
./ * Changes by John J. Programmer
```

is not processed by the UPDATE command when it creates the new source file, but it is written into the update log file.

Sequencing Output Records

The UPDATE command expects source files to have sequence numbers in columns 73 through 80. If you use the XEDIT subcommand SET SERIAL to sequence your files, the sequence numbers are usually written in columns 76 through 80; columns 73 through 75 contain a three-character identifier that is usually the first three characters of the filename. If you want an eight-character sequence number and you are editing the file, you must use the subcommand:

```
serial all
```

prior to issuing a FILE or SAVE subcommand. Or, you can create an UPDATE file with the single record:

```
./ S
```

and issue the UPDATE command to sequence the file.

If you use the UPDATE command with a file that has been sequenced using the default values of XEDIT, you must use the NOSEQ8 option. Otherwise, the UPDATE command cannot process your input file. The command:

```
update sample (noseq8
```

tells UPDATE to use only columns 76 through 80 when it looks for sequence numbers. Figure 12 shows the four files involved in a simple update.

The Source File, SAMPLE ASSEMBLE

```
SAMPLE   CSECT                                                          00000100
         USING SAMPLE,R12                                               00000200
         LR    R12,R15                                                  00000300
         ST    R14,SAVRET                                               00000400
         LINEDIT TEXT='PLEASE ENTER YOUR NAME'                          00000500
         RDTERM NAME                                                    00000600
         LINEDIT TEXT='PLEASE ENTER YOUR AGE'                           00000700
         RDTERM AGE                                                     00000800
         LINEDIT TEXT='HI, .........., YOU JUST TOLD ME YOU ARE .....',x00000900
               SUB=(CHARA,NAME,CHARA,AGE),RENT=NO                       00001000
         L     R14,SAVRET                                               00001100
         BR    R14                                                      00001200
         EJECT                                                          00001300
NAME     DC    CL130' '                                                 00001400
AGE      DC    CL130' '                                                 00001500
SAVRET   DC    F'0'                                                     00001600
         REGEQU                                                         00001700
         END                                                            00001800
```

The Update File, SAMPLE UPDATE

```
./ * REVISION BY DLC                                                    SAM00010
./ R 500                                                                SAM00020
         LINEDIT TEXT='WHAT''S YOUR NAME?',DOT=NO                       SAM00030
./ R 700 1000                                                           SAM00040
         LINEDIT TEXT='HI, .........., ENTER THE DOCNAME',             xSAM00050
               SUB=(CHARA,NAME)                                         SAM00060
         RDTERM NAME                                                    SAM00070
         MVC   DOCFN,NAME                                               SAM00080
         LA    1,PLIST                                                  SAM00090
         SVC 202                                                        SAM00100
         DC    AL4(ERROR)                                               SAM00110
RETURN   EQU   *                                                        SAM00120
./ I 1200                                                               SAM00130
ERROR    EQU   *                                                        SAM00140
         WRTERM 'FILE NOT FOUND'                                        SAM00150
         B     RETURN                                                   SAM00160
./ D 1500                                                               SAM00170
./ I 1600                                                               SAM00180
PLIST    DS    0D                                                       SAM00190
         DC    CL8'TYPE'                                                SAM00200
DOCFN    DC    CL8' '                                                   SAM00210
         DC    CL8'FILE'                                                SAM00220
         DC    CL8'A1'                                                  SAM00230
         DC    8X'FF'                                                   SAM00240
```

Figure 12 (Part 1 of 2). Updating Source Files with the UPDATE Command

The Record of Updates File, SAMPLE UPDLOG

```
UPDATING 'SAMPLE   ASSEMBLE A1' WITH 'SAMPLE   UPDATE   A1'           UPDATE LOG -- PAGE      1
        ./ * REVISION BY DLC
        ./ R 500
DELETING...              LINEDIT TEXT='PLEASE ENTER YOUR NAME'                          00000500
INSERTING...             LINEDIT TEXT='WHAT''S YOUR NAME?',DOT=NO                       ********
        ./ R 700 1000
DELETING...              LINEDIT TEXT='PLEASE ENTER YOUR AGE'                           00000700
                         RDTERM AGE                                                     00000800
                         LINEDIT TEXT='HI, ..........., YOU JUST TOLD ME YOU ARE .....',x00000900
                               SUB=(CHARA,NAME,CHARA,AGE),RENT=NO                       00001000
INSERTING...             LINEDIT TEXT='HI, .........., ENTER THE DOCNAME',             x********
                               SUB=(CHARA,NAME)                                         ********
                         RDTERM NAME                                                    ********
                         MVC    DOCFN,NAME                                              ********
                         LA     1,PLIST                                                 ********
                         SVC 202                                                        ********
                         DC     AL4(ERROR)                                              ********
                RETURN   EQU    *                                                       ********
        ./ I 1200
INSERTING...    ERROR    EQU    *                                                       ********
                         WRTERM 'FILE NOT FOUND'                                        ********
                         B      RETURN                                                  ********
        ./ D 1500
DELETING...     AGE      DC     CL130' '                                                00001500
        ./ I 1600
INSERTING...    PLIST    DS     0D                                                      ********
                         DC     CL8'TYPE'                                               ********
                DOCFN    DC     CL8' '                                                  ********
                         DC     CL8'FILE'                                               ********
                         DC     CL8'A1'                                                 ********
                         DC     8X'FF'                                                  ********
```

The Updated Output File, $SAMPLE ASSEMBLE

```
SAMPLE   CSECT                                                                          00000100
         USING SAMPLE,R12                                                               00000200
         LR    R12,R15                                                                  00000300
         ST    R14,SAVRET                                                               00000400
         LINEDIT TEXT='WHAT''S YOUR NAME?',DOT=NO                                       ********
         RDTERM NAME                                                                    00000600
         LINEDIT TEXT='HI, .........., ENTER THE DOCNAME',                             x********
               SUB=(CHARA,NAME)                                                         ********
         RDTERM NAME                                                                    ********
         MVC   DOCFN,NAME                                                               ********
         LA    1,PLIST                                                                  ********
         SVC 202                                                                        ********
         DC    AL4(ERROR)                                                               ********
RETURN   EQU   *                                                                        ********
         L     R14,SAVRET                                                               00001100
         BR    R14                                                                      00001200
ERROR    EQU   *                                                                        ********
         WRTERM 'FILE NOT FOUND'                                                        ********
         B     RETURN                                                                   ********
         EJECT                                                                          00001300
NAME     DC    CL130' '                                                                 00001400
SAVRET   DC    F'0'                                                                     00001600
PLIST    DS    0D                                                                       ********
         DC    CL8'TYPE'                                                                ********
DOCFN    DC    CL8' '                                                                   ********
         DC    CL8'FILE'                                                                ********
         DC    CL8'A1'                                                                  ********
         DC    8X'FF'                                                                   ********
         REGEQU                                                                         00001700
         END                                                                            00001800
```

Figure 12 (Part 2 of 2). Updating Source Files with the UPDATE Command

The INSERT and REPLACE statements allow you to control the numbering increment of records that you add to a source file. Notice, in Figure 12 on page 100 that inserted records have the character string '********' in columns 73 through 80. If you want sequence numbers on the inserted records, you must do two things:

1. Include a dollar sign ($) on the INSERT or REPLACE statement, optionally followed by operands indicating how the records should be sequenced.

2. Use the INC option on the UPDATE command line. If you use the CTL option, you do not have to specify the INC option. The CTL option is described below, under "Multiple Updates."

For example, to sequence the records added in Figure 12 on page 100 the control statements would appear as:

```
./ R  500 $
./ R  700 1000 $
./ I  1200 $
./ I  1600 $
```

and you would issue the UPDATE command:

```
update sample (inc
```

The UPDATE command sequences inserted records by increments of 10. If you want to control the numbering (for example, inserting more than 9 statements between two existing statements), you can specify an alternate sequencing scheme:

```
 ./ I  1800 $ 1805  5
.
.
.
```

Records introduced following this INSERT statement are numbered 00001805, 00001810, 00001815, and so on. (If the NOSEQ8 option is in effect, then the records would be xxx01805, xxx01810, and so on, where xxx is the three-character identifier used in columns 73 through 75.)

Multiple Updates

If you have several UPDATE files to apply to the same source, you may apply them in a series of UPDATE commands. For example, if you have updates named FICA UPDTUP1, FICA UPDTUP2, and FICA UPDTUP3 to apply to the source file FICA PLIOPT, you could do the following:

1. Update the source file with FICA UPDTUP1:

   ```
   update fica pliopt a fica updtup1
   ```

2. Update the source file produced by the above command with the FICA UPDTUP2:

```
update $fica pliopt a fica updtup2
```

3. Update the new source file with FICA UPDTUP3:

```
update $$fica pliopt a fica updtup3
```

This final UPDATE command produces the file $$$FICA PLIOPT, which is now the fully updated source file. This method is cumbersome, however, particularly if you have many updates to apply. They must be applied in a particular order. Therefore, the UPDATE command provides a multilevel update scheme, which you can use to apply many updates at one time, in a specified order.

To apply multilevel updates, you must have a control file, which by convention has a filetype of CNTRL and a filename that is the same as the source input file. Therefore, to apply the three update files to FICA PLIOPT, you should create a file named FICA CNTRL.

The Control File

A control file is actually a list. It does not contain any actual update control statements (INSERT, DELETE, and so on), but rather it indicates what update files should be applied, and in what order. In the case of a multilevel update, all the update files must have the same filename as the source file. Therefore, only the *filetypes* need be specified in the control file to uniquely identify the update file. In fact, since all your update files specified in a control file must have filetypes beginning with the characters UPDT, you need only specify the unique part of the filetype. The control file for FICA PLIOPT, named FICA CNTRL, may typically look like the following:

```
TEXT MACS PLILIB
FICA3 UP3
FICA2 UP2
FICA1 UP1
```

The first non-commentary record in the control file must be a MACS record. The second field in this record must be "MACS", and it may be followed by up to 29 macro library names (subject to the character limit of the line). Every record in the control file must have an "update level identifier." In this example, the update level identifiers are TEXT on the MACS record, FICA1 for the UP1 record, and so on. The update level identifier may have a maximum of five characters. See the "The STK Option" on page 111 for more details about the "update level identifier."

The UPDATE command only uses the MACS record and the update level identifier under special circumstances. These are described later under "The VMFASM EXEC Procedure" on page 109. For now, you only need to know that these things must be in a control file in order for the UPDATE command to execute properly.

Then, to update FICA PLIOPT, issue the UPDATE command as follows:

```
update fica pliopt (ctl
```

When you use the CTL option and you do not specify the name of a control file, the UPDATE command looks for a control file with the filetype of CNTRL and a filename the same as the source file. From the control file, it reads the filetypes of the updates to be applied. In this example, the UPDATE command searches for the file FICA UPDTUP1 and if found, applies the updates; then UPDATE searches for FICA UPDTUP2, and applies those updates, if any. Last it searches for FICA UPDTUP3, and applies those updates.

Notice that the updates are applied from the bottom of the control file, toward the top. This becomes important when an update is dependent on a previous update. For example, if you add some lines to a file in FICA UPDTUP1, then modify one of those lines in FICA UPDTUP2, it is important that UPDTUP1 was applied first.

Alternate Ways of Naming the Control Files

The example above, showing FICA CNTRL and UPDTxxxx files, illustrates a naming scheme using the UPDATE command defaults. You can override the defaults for the control file's filename and filetype.

For example, if you name a control file GROUPA CNTRL, you can specify the name of the control file on the UPDATE command line. Then to update FICA PLIOPT using the GROUPA CNTRL control file, issue the following UPDATE command:

```
update fica pliopt a groupa cntrl (ctl
```

AUX Files

The two levels of update processing shown so far may be adequate for your applications. There is, however, an additional level or step in the update structure that the VM/SP procedures use and that you may want to use also.

These techniques may be useful when you have more than one set of updates to apply to a source program. For example, you may have two groups of programmers who are working on different sets of changes for the same source file. Each group may create several update files and have a unique control file. When you combine these changes, you could create one control file or you can use what are known as auxiliary control files.

The updating structure for auxiliary control files is based on conventions for assigning filenames and filetypes. If a control file contains an entry that begins with the characters "AUX", the UPDATE command assumes that the file "fn AUXnnnn" contains a list of filetypes, not UPDATE control statements. For example, if the file SAMPLE ASSEMBLE is being updated with a control file that contains the record:

```
TEST1 AUXLIST
```

Then SAMPLE AUXLIST does not contain UPDATE control statements. It contains entries indicating the *filetypes* of the update files, all of which must have the same filename, SAMPLE.

Let's expand the example to see how this structure works. We have the source file, SAMPLE ASSEMBLE. The file SAMPLE CNTRL contains the entries:

```
TEXT MACS
3676 AUXLIST
```

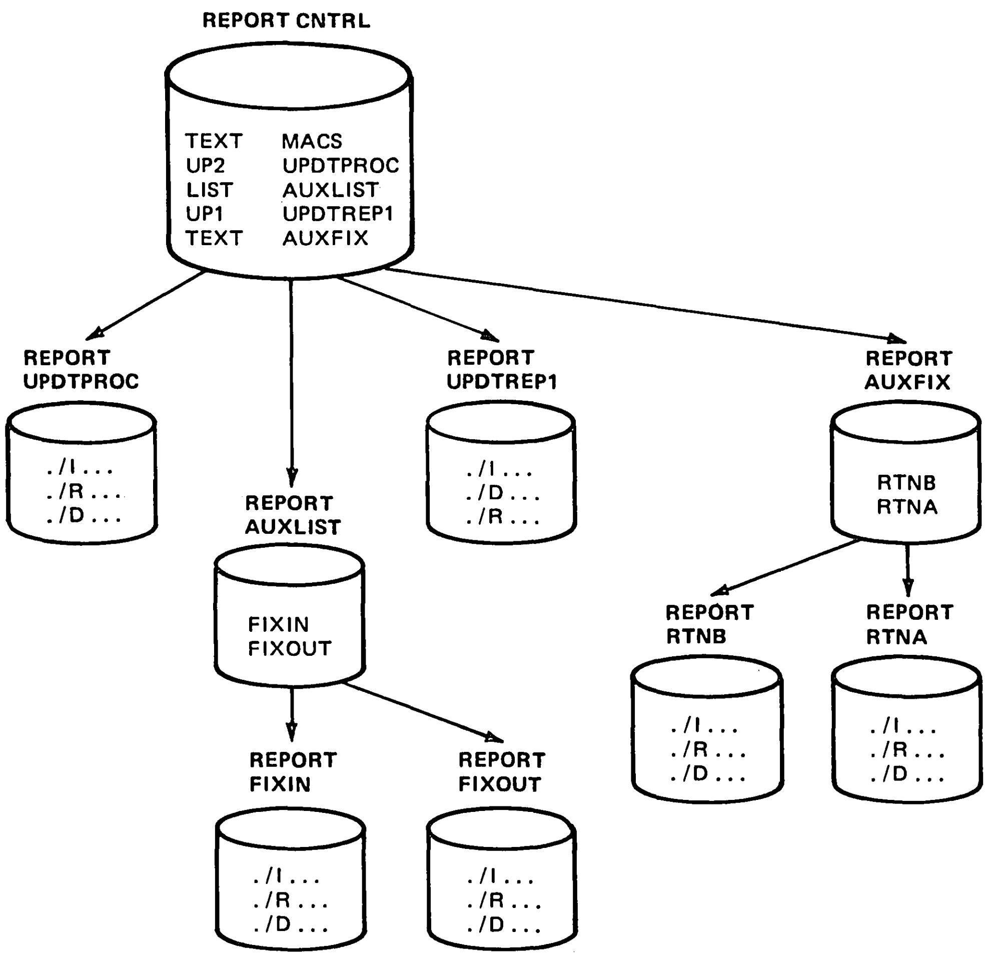
The file, SAMPLE AUXLIST may look like the following:

```
TEST1
FIXLOOP
BYPASS
```

The files:

```
SAMPLE TEST1
SAMPLE FIXLOOP
SAMPLE BYPASS
```

all contain UPDATE control statements (INSERT, DELETE, and so on) to be applied to the file SAMPLE ASSEMBLE. As with control file processing, the updates are applied from the bottom of the AUX file, so the updates in SAMPLE BYPASS are applied first, then the updates in SAMPLE FIXLOOP are applied, and so on. For an illustration of a set of update files, see Figure 13 on page 106.



```
update report assemble a (ctl)
UPDATING 'REPORT ASSEMBLE A1' WITH 'REPORT RTNA A1'.
UPDATING WITH 'REPORT RTNB A1'.
UPDATING WITH 'REPORT UPDTREP1 A1'.
UPDATING WITH 'REPORT FIXOUT A1'.
UPDATING WITH 'REPORT FIXIN A1'.
UPDATING WITH 'REPORT UPDTPROC A1'.
R;
```

Figure 13. An Update with a Control File

Since the updating scheme uses only filetypes to uniquely identify update files, it is possible to use the same control file to update different source input files. For example, issue the following command when using the control file REPORT CNTRL shown in Figure 13 on page 106:

```
update fica pliopt a report cntrl (ctl
```

The UPDATE command begins searching for updates to apply to FICA PLIOPT, based on the entries in REPORT CNTRL. It searches for FICA AUXFIX, which may contain entries pointing to update files; then it searches for FICA UPDTREP1, and so on.

As long as all updates and auxiliary files associated with a source file have the same filename as the source file, the updates are uniquely identifiable. Therefore, the same control file can be used to update various source files. VM/SP takes advantage of this capability in its own updating procedures. By maintaining strict naming conventions, updates to various CP and CMS modules are easily controlled and identified.

A control file may point to many AUX files in addition to many UPDT files. You can modify a control file when you want to control which updates are applied to a program. You may have several control files, and specify the name of the control file you want to use on the UPDATE command line. There is a lot of flexibility in the UPDATE command processing. You can implement procedures and conventions for your individual applications.

Multiple Updates with XEDIT

The XEDIT CTL option creates multiple updates to a source file. First, create a control file listing the updates to be applied to a source file. Initially, you might have only the MACS record and one UPDATE filetype specified. For example, you can create a file called FICA CNTRL that contains:

```
TEXT MACS PLILIB
FICA1 UPDTUP1
```

Next, specify the control file name that you have created after the XEDIT CTL option. For example:

```
xedit fica pliopt (ctl fica
```

The editor searches for an update file called FICA UPDTUP1 and applies all updates contained in this file. If the update file does not exist, XEDIT creates a file called FICA UPDTUP1 which will contain all changes made to the source file during the editing session in addition to the required control statements.

If you wish to add another level of updates to your source file, insert a new update filetype in your control file after the MACS record, for example:

```
TEXT MACS PLILIB
FICA2 UPDTUP2
FICA1 UPDTUP1
```

Then, XEDIT your source file again, specifying the CTL option, for example:

```
xedit fica pliopt (ctl fica
```

XEDIT applies all updates contained in FICA UPDTUP1 to the source file FICA PLIOPT. After the resulting file is displayed, any additional updates and the necessary control statements are automatically inserted in another update file called FICA UPDTUP2, consistent with control file processing from the bottom up.

Auxiliary control files can also be used with XEDIT. You can make your control file point to AUX files that contain the filetypes of the actual update files, or you can combine AUX files and update files in a single control file. XEDIT begins applying updates from the bottom up in the control file and references the AUX files indicated. Any updates to the source file produced during the editing session are inserted in the topmost update filetype specified in either the control file or in the last AUX file encountered using the 'bottom up' processing rule. More information about the XEDIT CTL option can be found in the *VM/SP System Product Editor Command and Macro Reference.*

Preferred Level Updating

There may exist more than one version of an update, each applicable to different versions of the same module. For example, you may need one version of an update for an unmodified base source module and another version of that update if that module has been modified by a licensed program. The AUX file used to update a particular module must then be selected based on whether or not a licensed program modifies that module. The AUX files listing the updates applicable to modules modified by a licensed program are called "preferred AUX files" because they must be used if they exist rather than the mutually exclusive updates applicable to unmodified modules. Using this preferred AUX file concept, every module in a component can be assembled using the one CNTRL file applicable to a user's configuration.

A single AUX file entry in a CNTRL file can specify more than one filetype. The first filetype indicates a file that UPDATE uses only on one condition: the files that the second and subsequent filetypes indicate do not exist. If they do exist, this AUX file entry is ignored and no updating is done. The files that the second and subsequent filetypes indicate are preferred because UPDATE does not use the file that the first filetype indicates. Usually, the preferred files appear later in the CNTRL file in a format that causes them to be used for updating.

UPDATE scans each CNTRL file entry until a preferred filetype is found, until there are no more filetypes on the entry, or until a comment is found. (A character string less than four or more than eight characters is assumed to be a comment.)

The VMFASM EXEC Procedure

If you are an assembler language programmer and you are using the UPDATE command to update source programs you may want to use the VMFASM EXEC procedure. VMFASM is a VM/SP update procedure. It invokes the UPDATE command and uses the ASSEMBLE command to assemble the updated source file.

If you are not an assembler language programmer, you may wish to create an EXEC similar to VMFASM that calls one of the language compilers to compile an updated source file, instead of calling the assembler.

When you use VMFASM, you specify the source filename, the filename of the control file, and optionally, parameters for the assembler. (The control file for VMFASM must have a filetype of CNTRL). For example, if you use the file GENERAL CNTRL to update SAMPLE ASSEMBLE, you enter the command line:

```
vmfasm sample general
```

The VMFASM EXEC uses the MACS card and the update level identifiers in the control file. It reads the MACS card to determine which macro libraries (MACLIBs) should be searched by the assembler. Then VMFASM issues the GLOBAL MACLIB command specifying the MACLIBs you name on the MACS card.

VMFASM uses the update level identifier to name the output text file produced by the assembly. If the update level identifier of the most recent update file (the last one located and applied) is anything other than TEXT, the update level identifier is prefixed with the characters TXT to form the filetype. For example, if the file GENERAL CNTRL contains the records:

```
TEXT MACS CMSLIB MYLIB OSMACRO
UP2 FIX2
UP1 FIX1
TEXT AUXLIST
```

and updates the file SAMPLE ASSEMBLE, then:

- If the file SAMPLE UPDTFIX2 is found and the updates applied, VMFASM names the output text deck SAMPLE TXTUP2.
- If the file SAMPLE UPDTFIX1 is found and the updates applied but no SAMPLE UPDTFIX2 is found, the text deck is named SAMPLE TXTUP1.
- If the file SAMPLE AUXLIST is found but no SAMPLE UPDTFIX1 or SAMPLE UPDTFIX2 files are found, the text deck is named SAMPLE TEXT.
- If no files are found, the update level identifier on the MACS card is used and the text deck is named SAMPLE TEXT.

The new fn TEXT or fn TXTxxxxx resides on the A-disk. Because the UPDATE command works from the bottom of a control file toward the top, it is logical that the text filename be taken from the identifier of the last update applied.

The VMFASM EXEC does not produce an updated source file, but leaves the original source intact.

VMFASM produces two output files:

- A printed output listing that shows update activity
- The text file that contains the update log as well as the actual object code.

If you use the CMS LOAD command to load a text file produced by VMFASM, records from the update log are flagged as invalid, but the LOAD operation is not impaired.

Updating EXECs and Macros

If you wish to use the update facility to track changes to EXECs or macros written for the System Product Interpreter, you need to use the EXECUPDT command. The EXECUPDT command applies updates to an EXEC source file (using the UPDATE command) and removes the sequence numbers from the updated file to produce an executable version of the file. Using EXECUPDT is very similar to using the VMFASM EXEC to apply updates to an assembler language source and to assemble it.

Source files for the EXECUPDT command are fixed-length, 80-character files with sequence numbers just like those for assembler language or COBOL. The filetype of the EXEC source file has a '$' prefixed to the normal filetype. For example, SAMPLE $EXEC could be the source for an EXEC procedure and READY $XEDIT could be a source file for an XEDIT macro.

Updates to the EXEC source are created using XEDIT in the same manner as updates to programs in other languages. To apply the updates to the source, use the EXECUPDT command. For a single level update, issue the command:

```
execupdt sample exec
```

Note that the '$' in the filetype is not included in the filetype specified on the EXECUPDT command. To do a multi-level update, you may use the CTL option of EXECUPDT. For example:

```
execupdt sample exec (ctl general
```

The STK Option

If you are interested in writing your own EXEC procedure to invoke the UPDATE command, you may wish to use the STK option. The STK (stack) option is valid only with the CTL option and is meaningful only when the UPDATE command is invoked within an EXEC procedure.

When the STK option is specified, UPDATE stacks the following data lines in the console stack:

```
first line:  * update level identifier
second line: * library list from MACS record
```

The update level identifier is the identifier of the most recent update that was found and applied.

For example, an EXEC 2 EXEC that invokes the UPDATE command and then the ASSEMBLE command may contain the lines:

```
&TRACE ALL
UPDATE &1 ASSEMBLE * &2 CNTRL * (STK CTL
&READ VARS &STAR &TX
&READ VARS &STAR &LIB1 &LIB2 &LIB3 &LIB4 &LIB5 &LIB6 &LIB7 &LIB8
GLOBAL MACLIB &LIB1 &LIB2 &LIB3 &LIB4 &LIB5 &LIB6 &LIB7 &LIB8
&IF &TX NE TEXT FILEDEF TEXT DISK &1 TXT&TX A1
ASSEMBLE &1 &3 &4 &5 &6 &7 &8 &9 &10
ERASE $&1 ASSEMBLE
```

Below is a System Product Interpreter program that invokes the UPDATE command and then the ASSEMBLE command:

```
 /* Sample System Product Interpreter program to */
 /* Update and Assemble a source program         */
 trace a
 parse arg filename cntrlfile options
'UPDATE' filename 'ASSEMBLE *' cntrlfile 'cntrl * (STK CTL'
 parse pull star tx
 parse pull star lib1 lib2 lib3 lib4 lib5 lib6 lib7 lib8
'GLOBAL MACLIB' lib1 lib2 lib3 lib4 lib5 lib6 lib7 lib8
 if tx ¬= TEXT then 'FILEDEF TEXT DISK' filename 'TXT'tx 'A1'
'ASSEMBLE $'filename options
'ERASE  $'filename 'ASSEMBLE'
```

If the EXEC that you use is named UPASM EXEC, it is invoked with the line:

```
upasm fica fica (print noxref
```

and the file FICA CNTRL contains:

```
MAC MACS CMSLIB OSMACRO MYTEST
FIX1  UPDTFIX
LIST  AUXLIST
```

then the EXEC 2 EXEC executes the following commands:

```
UPDATE FICA ASSEMBLE * FICA CNTRL * (STK CTL
GLOBAL MACLIB CMSLIB OSMACRO MYTEST
FILEDEF TEXT DISK FICA TXTFIX1 A1
ASSEMBLE $FICA (PRINT NOXREF
ERASE $FICA ASSEMBLE
```

The System Product Interpreter program executes the following:

```
/* Update FICA ASSEMBLE using FICA CNTRL */
'UPDATE FICA ASSEMBLE * FICA CNTRL * (STK CTL'
'GLOBAL MACLIB CMSLIB OSMACRO MYTEST'
'FILEDEF TEXT DISK FICA TXTFIX1 A1'
'ASSEMBLE $FICA (PRINT NOXREF'
'ERASE $FICA ASSEMBLE'
```

The above examples assume that the update file FICA UPDTFIX was found and applied.

Chapter 7. Developing Commands and Message Files

Using the Parsing Facility

The CMS parsing facility parses and translates command arguments. Your programs can use the parsing facility to see if the user specifies the proper arguments on invocation and to see what the arguments are. For a list of CMS commands that use the parsing facility, see "Supported CMS Commands" on page 116.

Advantages of the Parsing Facility

When you use the parsing facility, programming commands is simpler because:

- The parsing facility detects invalid command arguments.
- All keyword abbreviations are expanded for you.
- Command syntax is defined separately from your program and can be translated into different national languages.
- When a national language is in use, keywords in that language are translated into the language recognized by your program.
- You do not have to write scanning code.
- The address and length of each token is provided.
- Validation codes are provided to identify the type of each token.

Advantages of DLCS

To use the parsing facility, you must define command syntax in a special language, the definition language for command syntax (DLCS).

Keep the DLCS definitions in CMS files. A file can contain more than one DLCS definition. The parsing facility parses a specified command by checking whether all operands, options, keywords, and so on, 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:

- Syntax checking is unnecessary in your program.
- If you want to invoke your program in another national language, you just have to modify your DLCS file.

Refer to "Coding Your Own Command Syntax with DLCS" on page 117 for details on using DLCS.

Overview

To have the parsing facility do this checking, do the following:

1. Write DLCS statements.
2. Check for any DLCS coding errors using the CHECK option of the CONVERT COMMANDS command.
3. Issue the CONVERT COMMANDS command to put your syntax file into a machine readable form the parsing facility can use.
4. Issue the SET LANGUAGE command to enable the user's DLCS definitions.
5. Issue the PARSECMD command from a REXX program or EXEC 2 EXEC or the PARSECMD macro from an assembler program to invoke the parsing facility and to obtain the parsed and translated parameter lists.

Coding DLCS Statements

To show how DLCS statements work, here is a standard CMS command string format:

```
command_name  [operands...] [(options...]
```

DLCS has the following statements:

:CMD	for a command name
:OPR	for an operand
:OPT	for an option

A few other statements you can use in DLCS include:

:SYN	to define synonyms
:KW.n	for command name modifiers
:*	to specify comments

For example, the RDRLIST command has the following format:

```
RDRLIST [( [PROFile fn] [Append] [)]]
```

Here is how the syntax for RDRLIST is coded in DLCS:

```
:CMD D9K.RDRLIST RDRLIST RDRLIST 4 :;

  :SYN RLIST 2 :;
  :OPT KWL( <PROFILE 4> ) FCN(FN) :;
  :OPT KWL( <APPEND 1> ) :;
```

The section, "Coding Your Own Command Syntax with DLCS" on page 117, describes DLCS statements in detail.

Converting Your DLCS File

When you are ready to use the DLCS file, you first have to convert the DLCS file into an machine readable form the parsing facility can use. Use the CONVERT COMMANDS command to do this.

You can use the CHECK option of CONVERT COMMANDS to make sure your DLCS syntax descriptions are correct. In addition, you can issue CONVERT COMMANDS with the CHECK option while you XEDIT the DLCS file to help remove errors. Next, use the SET LANGUAGE command to put the new DLCS definitions into effect.

Refer to the *VM/SP CMS Command Reference* for a complete description of CONVERT COMMANDS and SET LANGUAGE.

Setting Command Name Synonyms and Translations

Use the SET TRANSLATE command to set user translation synonyms, user translations, system translation synonyms, and system translations on or off. Use the QUERY TRANSLATE command to display the contents of the system synonym tables, system translate tables, user synonym tables, and user translate tables.

These commands work similarly to the CMS SYNONYM and QUERY SYNONYM commands.

(Refer to the *VM/SP CMS Command Reference* for descriptions of SET TRANSLATE, QUERY TRANSLATE, SYNONYM, and QUERY SYNONYM.)

Invoking the Parsing Facility

You can invoke the parsing facility in two ways:

1. From EXEC 2 EXECs or REXX programs, use the PARSECMD command.

   The PARSECMD command uses the EXECCOMM interface and creates EXEC variables that describe the translated command string. See Figure 14 on page 135 for an example using the PARSECMD command.

   Refer to the *VM/SP CMS Command Reference* for a description of the PARSECMD command.

2. From assembler programs, use the PARSECMD macro.

   The PARSECMD macro call should be in the beginning of the program. Upon return from the parsing facility, the syntax of the command is verified and detailed information on the translated command string is available. See Figure 16 on page 138 for an example using the PARSECMD macro.

   Refer to the *VM/SP CMS Macros and Functions Reference* for a description of the PARSECMD macro.

Command keywords are uppercased according to the national language uppercase table for the active application. If one is not found, the CMS national language table is used.

Supported CMS Commands

The following commands use the parsing facility:

ACCESS	GET	POP	SIZE
ALARM	HELP	POSITION	SORT
CLEAR	HELPCONV[3]	PRINT	STATE
COMPARE	HIDE	PUNCH	SVCTRACE
CONVERT	IDENTIFY	PUT	SYNONYM
COPYFILE	LANGGEN	QUERY	TAPE
CURSOR	LANGMERG	RDR	TELL
DEFAULTS	LISTFILE	RDRLIST	TXTLIB
DEFINE	MACLIST	READCARD	TYPE
DELETE	MAXIMIZE	RECEIVE	UPDATE
DISK	MINIMIZE	REFRESH	WAITREAD
DROP	MODMAP	RELEASE	WAITT
ERASE	MOREHELP	RENAME	WRITE
EXECDROP	NAMES	RESERVE	XEDIT[4]
EXECLOAD	NOTE	RESTORE	XMITMSG
EXECMAP	NUCXDROP	ROUTE	
EXECSTAT[3]	NUCXLOAD	SCROLL	
FILELIST	NUCXMAP	SENDFILE	
FORMAT	PARSECMD	SET	
GENMSG	PEEK	SHOW	

[3] These commands do not have parameters requiring translation, but the command name itself can be translated.

[4] XEDIT subcommands are not supported.

Coding Your Own Command Syntax with DLCS

Rules to Remember

Some rules to remember while coding in DLCS are:

- Use special characters : , < > and ′ in your data tokens (keyword names, function names, or function values) only if they are enclosed in single quotes. The quotes are not counted as part of the token.
- Do not use lowercase characters to specify your keyword names, function names, or function values. Specify these exactly as they appear after the command line is uppercased by the system at execution time with the language in effect.
- Only the first 72 characters of any line of the DLCS file are used. Any characters beyond 72 are ignored. You can use as many blanks as you want between tokens, and you can continue DLCS statements on the following line.
- Only one system and one user DLCS file for an application can be made active at any time. When both a user file and a system file are active, the definitions in the user file override the definitions in the system file. If no syntax definition is found in the user file, the definition in the system file is used.
- Your DLCS file must be merged with your user file for the application you currently have. You can have only one user table; therefore, if you have another command or receive a command from someone, you have to merge it with the other commands in the user table. (For example, if you want the CMS search order to find your command, define the command in a DMS file.)
- You can define the translation of some keywords to be the same as the keyword the command recognizes. For more information on translation, see the *VM/SP CMS User's Guide.*

Defining Your Command

Define each command as follows:

1. Start with a CMD statement to specify the name of the command and its national language equivalent.
2. Define any synonyms using SYN statements immediately following the CMD statement.
3. Define a two word command using the first word as the command name and using the KW.1 statement to define the second word. If the command is a three word command, use the KW.2 statement to define the third word. (The second and third words are command name

modifiers.) You can also have a four word command, a five word command, etc.

4. Define the syntax for the command with zero or more OPR statements followed by zero or more OPT statements.

5. Use ':;' to specify the end of a statement.

SPECIFYING THE APPLICATION AND NATIONAL LANGUAGE

DLCS Statement: Use the DLCS statement to define the application where commands in the DLCS file are parsed, to specify whether the commands are system or user commands, and to specify the national language for the file.

The format of the DLCS statement is:

```
:DLCS  applid  System|User  langid  :;
```

where:

applid
is an application identifier. It must be three alphanumeric characters, and the first character must be alphabetic (e.g., DMS, DMK, OFS, AGW, DKK, etc...).

System|User
specifies whether the file contains system or user syntax definition statements. (Only the first letter is significant.)

langid
is the identifier for the language you are working in. It must be one to five alphanumeric characters (e.g., FRNCH, AMENG, etc...).

Notes:

1. *The DLCS statement must be the first non-comment statement in the DLCS file, and it must be the only DLCS statement in the file.*

2. *The CMS command search order uses translations and translation synonyms defined in DLCS files with an application identifier of DMS.*

3. *The DLCS statement determines the filename and filetype of the output files.*

DEFINING THE COMMAND NAME, SYNONYMS, AND MODIFIERS

CMD Statement: Use the CMD statement to define the name of a command as the system sees it and as the language sees it.

The format of the CMD statement is:

:CMD *unique-id* *sl-name* [*nl-name* *n*] **:;**

where:

unique-id
: identifies the syntax definition for the command within the DLCS file. This is required, and it must be unique for each syntax definition. When you invoke the parsing facility, *unique-id* is matched to the one you specify in the PARSECMD.

  unique-id is any combination of up to 16 characters. For quick access to the syntax definitions, the first one or two characters are used as an index. If the first two characters of *unique-id* are valid hexadecimal digits, their value is used as the index. Otherwise, the EBCDIC value of the first character is used. For example, *unique-id*s D9xxx and Rxxx both have the same index value of 217. CMS can find syntax definitions faster if you use as many of the 256 index values as possible.

sl-name
: is the command name as CMS sees it.

nl-name
: is the command name as a national language user sees it. Defaults to *sl-name.*

n
: is the minimum number of characters that must be entered for *nl-name* to be accepted. Defaults to the length of *sl-name.*

Notes:

1. *A new command syntax begins each time a CMD statement is encountered.*

2. *All uniqueids used for IBM commands have a period as the fourth character. Do not use a period as the fourth character in the the uniqueid for your own commands.*

3. *A uniqueid of all blanks is reserved to let you define more than one translation for a command. When this uniqueid is found, no syntax information is stored. You can only code the :CMD and :SYN statements in this case.*

4. *The minimum length for abbreviations of command name translations cannot be more than eight or HELP does not recognize them.*

5. *nl-name is only used by the CMS search order if the application identifier of this DLCS file is DMS.*

6. *The SET TRANSLATE command enables or disables nl-name.*

7. *If SET ABBREV OFF is in effect, you must use the full nl-name.*

SYN Statement: Use the SYN statement to define translation synonyms for the command name defined on the :CMD statement.

The format of the SYN statement is:

```
:SYN   newname1  n1  [newname2  n2 ]  ...   :;
```

where:

newname
: is the synonym you are assigning to the command name.

n
: is the minimum number of characters you must enter for the synonym to be accepted by CMS.

Notes:

1. *The SYN statement is valid only for the first word of a command name (not the command name modifiers).*

2. *All of the SYN statements for a command must immediately follow the CMD statement.*

3. *Only SYN statements defined in a DLCS file with an application identifier of DMS are used by the CMS command search order.*

4. *The SET TRANSLATE command enables and disables translation synonyms defined by the :SYN statement.*

5. *Using multiple names on a single SYN statement has the same effect as specifying a single name on many SYN statements. Order is not important.*

6. *If SET ABBREV OFF is in effect, you must use the full newname1.*

KW.n Statement: Use the KW.n statement to define command name modifiers keywords that modify the syntax used for parsing the remaining parameters. For example, a command to manipulate a simple data base can require different operands-- a filename for an open request, an option for a close request, and other operands for update requests. The KW.n statement lets you define a different syntax for each.

The format of the KW.n statement is:

```
:KW.n  sl-name  sl-abbrev  [nl-name  nl-abbrev]  :;
```

where:

n
: is the number of the level. It defines the *n*th modifier after the command name.

sl-name
: is the name as the command sees it.

sl-abbrev
: is the minimum number of characters that must be entered for *sl-name* to be accepted by CMS.

nl-name
: is the name as the national language user enters it. Defaults to *sl-name.*

nl-abbrev
: is the minimum number of characters that must be entered for *nl-name* to be accepted by CMS. Defaults to *sl-abbrev.*

Use the following form of the :KW.n statement to indicate that a string of characters not defined by any :KW.n statement is accepted as an arbitrary modifier.

```
:KW.n  :;
```

Note: This form may not be used as the first :KW.n statement on a level, and only applies to :KW.n statements on the same level. No further syntax information may follow this statement, that is, no :OPR, :OPT, or :KW.n statements with a larger value for n. When the parsing facility finds an arbitrary modifier it will process that remainder of the argument string as one text string.

Example:

Suppose the format of a database command is:

```
DATABASE   OPEN   filename
           UPDATE ROW      row number
           UPDATE COLUMN   column-name
           CLOSE  [(REPLACE [)]]
```

When the DLCS for this command is coded, instead of defining OPEN, UPDATE, and CLOSE as operand keywords, they are coded as modifiers (because they modify the syntax) using the KW.n statement. Because each is the first modifier following the command name, the modifier level (the n part of KW.n) is 1. In this way, you can define a command with many modifiers at the same level. You can define the remaining operands and options differently for OPEN, UPDATE, and CLOSE. The DLCS definition so far is:

```
:CMD DATABASE DATABASE :;
  :KW.1 OPEN 4 :;
    :OPR FCN(FN) :;
  :KW.1 UPDATE :;
  :KW.1 CLOSE 5 :;
    :OPT KWL(<REPLACE 3>) :;
```

The keywords ROW and COLUMN can only follow UPDATE, and they modify the syntax further. Each is the second modifier after the command name, so they are coded as a second level modifier following UPDATE. Each KW.2 statement on this second level may be followed by either a third level or operand and option definitions, and so on. These KW.2 statements nest after the previous KW.1 statement so that ROW or COLUMN are only recognized after UPDATE. The complete DLCS definition for the database command is:

```
:CMD DATABASE DATABASE :;   :* A sample syntax
  :KW.1 OPEN 4 :;
      :OPR FCN(FN) :;   :* filename
  :KW.1 UPDATE :;
    :KW.2 ROW 3 :;
      :OPR FCN(PINTEGER) :;   :* row-number
    :KW.2 COLUMN 3 :;
      :OPR FCN(STRING) :;   :* column-name
  :KW.1 CLOSE 5 :;
      :OPT KWL(<REPLACE 3>) :;
```

DEFINING OPERANDS

OPR Statement: Use the OPR statement to define the syntax of each operand of the command.

The format of the OPR statement is:

```
:OPR  KWL( kwdef1  [kwdef2  ...] )
                [ OPTIONAL | STOP ]   [ REPEAT ]  :;

:OPR  FCN( fcndef1  [fcndef2  ...] )
                [ OPTIONAL | STOP ]   [ REPEAT ]  :;

:OPR  KWL( kwdef1  [kwdef2  ...] )
                FCN( fcndef1  [fcndef2  ...] )
                [ OPTIONAL | STOP ]   [ REPEAT ]  :;
```

where:

KWL
: defines the keyword when an operand (or option when defining an option) is defined to be a keyword.

FCN
: defines functions to be used to validate the value of an operand.

KWL FCN
: defines a keyword-value pair using the kwdef and fcndef expressions. The kwdef and fcndef expressions are defined on pages 124 and 125.

OPTIONAL
: indicates the operand can optionally be specified.

STOP
: specifies that if the operand is not specified then parsing of the operands stops at that point and no more operands can be specified.

REPEAT
: indicates the operand can be specified one or more times.

Notes:

1. *Specify OPR statements in the order the operands are specified on the command.*

2. *Specify the OPR statement after the CMD statement and present SYN statements or after appropriate KW.n statements.*

3. *If both OPTIONAL (or STOP) and REPEAT are specified, the operand can be specified zero or more times.*

4. *If no options are specified, the operand is a required operand that can be specified only once.*

DEFINING OPTIONS

OPT Statement: Use the OPT statement to define the syntax of the options for the command.

The format of the OPT statement is:

```
:OPT  KWL( kwdef1  [kwdef2  ...] )   :;

:OPT  KWL( kwdef1  [kwdef2  ...] )
                   FCN( fcndef1  [fcndef2   ...] )   :;
```

where:

KWL
: defines the keyword when an option (or operand when defining an operand) is defined to be a keyword.

KWL FCN
: defines a keyword-value pair using the kwdef and fcndef expressions. The kwdef and fcndef expressions are defined below.

Note: OPT statements must follow the last OPR statement, if any were used, for that command. Order of the OPT statements is not important.

kwdef EXPRESSION

The format of kwdef is:

```
[  < sl-name  sl-abbrev  [nl-name  nl-abbrev] >  ]
```

where:

sl-name
: is the keyword known by your command.

sl-abbrev
: is the minimum number of characters that must be entered for *sl-name* to be accepted.

nl-name
: is the keyword known by a national language user. Defaults to *sl-name.*

nl-abbrev
: is the minimum number of characters that must be entered for *nl-name* to be accepted. Defaults to *sl-abbrev.*

fcndef EXPRESSION

fcndef can be any one of the system functions listed below. In addition, fcndef can be a user function. See "User Functions" on page 127 for more information.

System Functions

Syntax	**Description**
ALPHANUM	any alphanumeric string
APPLID	any three character alphanumeric string with the first alphabetic
CHAR	any single nonblank character
CUU	any hex number between 001 and FFF (assumes leading zeros)
DIGITS	any unsigned number made up of digits 0-9
FN	(filename) any string with the following characters: A-Z,a-z,0-9,$,#,@,+,-,:, and _
FT	(filetype) any string with the following characters: A-Z,a-z,0-9,$,#,@,+,-,:, and _
FM	(filemode) first character: A-Z, a-z; optional second character: 0-6
EFN	same as FN with '*' or '%' also a valid character
EFT	same as FT with '*' or '%' also a valid character
EXECNAME	any string that does not contain the following characters: =,*,(,),' ', and X'FF'

EXECTYPE	any string that does not contain the following characters: =,*,(,),' ', and X'FF'
HEX	any hexadecimal number
INTEGER	any decimal whole number (can have + or - signs)
NINTEGER	any decimal negative whole number
PINTEGER	any decimal positive whole number (can have + sign)
MODE	any alphabetic character
STRING	any nonblank character string
TEXT	any character string
INVALID	no valid values

Notes:

1. *You can specify function definitions with a subset of valid values. Only items in the subset are valid. For example, if you specify STRING(MONDAY, TUESDAY, WEDNESDAY), MONDAY, TUESDAY, and WEDNESDAY are the only valid values.*

2. *If a list of functions is specified for* ***fcndef****, the parsing facility validates an operand or option value with the functions in the order they are specified. The first function the value is valid for determines the validation code of the value in PVCENTRY. Refer to the VM/SP CMS Macros and Functions Reference for more information on the PVCENTRY macro.*

3. *Input to the parsing facility is uppercased according to current language before it is provided to system or user functions for validation.*

4. *If a value is not valid according to any of the functions in the list, the first one is used to determine which message, if any, is issued. If an error message based on the first function is not appropriate, place the INVALID function first in the list. For example:*

   ```
   :OPR FCN(INVALID, INTEGER(2,4,6), MODE) :;
   ```

   The invalid function never accepts a value as valid, but a general error message is issued when a value is not valid according to the rest of the functions in the list.

5. *Because some functions will validate tokens that are also valid for other functions, you should be careful to list the most restrictive functions first. For example, an operand defined as:*

   ```
   :OPR FCN(STRING, DIGITS, FN):;
   ```

will always be validated as a string, while the syntax:

```
:OPR FCN(FN) REPEAT:;
:OPR FCN(DIGITS):;
```

can never be satisfied because the required digits operand will be validated as part of the list of filenames.

6. *The TEXT function cannot be specified in a list with any other function.*

User Functions

In addition to the system functions listed above, you can also make your own functions for the parser to use to check if a token is valid. For instance, you could make a function VOWEL that considers only alphabetic characters A,E,I,O and U valid.

After you make your program for your function, assemble it, load it with the RLDSAVE option, and use the GENMOD command. Then install the MODULE file of this assembled program as a nucleus extension. Next, include the name of your function in the DLCS for your command exactly as you would any other function. The function is invoked by the parsing facility with an SVC 202. The entry point name of the module must be the same as the function name (fcndef) in your DLCS file. Your function is passed the following parameters:

- An eight byte area containing the function name
- token-addr: a fullword containing the address of the token to be validated. The token is already uppercased according to the current language.
- token-length: a fullword containing the number of characters in the token.
- validation code: a byte containing the number interpreted by the parser as the validation code of the user function. If the token is valid, this field should be set by the user function. Upon return from the parsing facility, you can check this validation code to see if your token is valid.

On entry to the program, R1 contains the address of the control block containing the parameters described above. Use the assembler macro PARSERUF to generate a mapping of this control block.

Your program must pass back a return code in R15 that determines the outcome of the function. A return code of zero specifies the token was valid; a non-zero return code specifies the token was not valid. You can use any non-zero return code except -3; this return code would be interpreted to mean the function did not exist.

Notes:

1. *User functions do not override system functions with the same name (system functions come first in the search order).*

2. *When you use CONVERT COMMANDS to process your DLCS file, specify the ALL or USER options for user functions to be accepted.*

3. *When coding user functions in your CDSL file, you can enclose specific values in parentheses as you can with any system functions and only those values are accepted.*

DEFINING ROUTINES AND KEYWORDS

Note: The :RTN and :KWD statements are reserved for IBM use. You may not use them in writing your own commands in DLCS. They are only shown here so that if you need to make your own translation of CMS commands you can do so without introducing errors into the syntax or its definition.

RTN Statements: Use the RTN statement to define the routine responsible for parsing the command.

The format of the RTN statement is:

```
:RTN    routine-name    :;
```

where:

routine-name
is a CMS defined name.

Notes:

1. *When the :RTN and :KWD statements are used, they replace (and are nutually exclusive with) the :OPR and :OPT statements. There is one :RTN statement followed by any number of :KWD statements.*

2. *When you are translating a CMS command that uses routine parsing, you should only changed the nl-name and nl-abbr fields on the :KWD statement. You must not add or delete :KWD statements or change the routine and system language names.*

KWD Statements: Use the KWD statement to define the keywords that the command contains for translation purposes.

The format of the KWD statement is:

```
:KWD  sl-name  sl-abbrev  [ nl-name  nl-abbrev ]  :;
```

where:

sl-name
: is the CMS defined keyword name.

sl-abbrev
: is the minimum number of characters that must be entered for *sl-name* to be accepted by CMS.

nl-name
: is the keyword as a national language user enters it.

nl-abbrev
: is the minimum number of characters that must be entered for *nl-name* to be accepted by CMS.

Notes:

1. *When the :RTN and :KWD statements are used, they replace (and are nutually exclusive with) the :OPR and :OPT statements. There is one :RTN statement followed by any number of :KWD statements.*

2. *When you are translating a CMS command that uses routine parsing, you should only changed the nl-name and nl-abbr fields on the :KWD statement. You must not add or delete :KWD statements or change the routine and system language names.*

Writing Comments

Comments: Use the characters :* to specify that a line or the remaining characters of a line are to be ignored. Use this to put comments and explanations in your DLCS file.

The format of a comment is:

```
[ DLCS statements or parts of a statement ]  :*  comment
```

where:

comment
is any comment

Creating a DLCS File

For examples of creating a DLCS file, see "Creating a DLCS File" on page 131 and "Creating a DLCS File with National Language Translations" on page 132.

What the Parser Does Not Flag

1. The parser **does not** flag the following situations:

   - Dependent options and operands. The MAP operand of the MACLIB command gives an example. Refer to the *VM/SP CMS Command Reference.*

   - Mutually exclusive options or operands. This is where you have a pair of operands or options. You must specify one or the other -- you cannot specify neither or both. The ACK and NOACK operands of the NOTE command give an example. Refer to the *VM/SP CMS Command Reference.* Most commands that have mutually exclusive options or operands ignore the condition and use the last operand or option you specify.

2. Some IBM supplied commands also use the RTN and KWD statements for special purposes. Do not use these statements for your own commands.

DBCS and the Parsing Facility

This section lists rules to remember when the current language is a double-byte character set (DBCS) language.

In DLCS and CONVERT COMMANDS

- You can use DBCS characters only in keyword, modifier, and command names.

- You can mix single byte and DBCS characters in a name in the DLCS, but CONVERT COMMANDS only recognizes single byte characters as DLCS delimiters.

- Shift-out and shift-in characters are always recognized as DBCS delimiters in a DLCS definition regardless of the current language.

- A double byte character is treated as a single logical character. When you specify the minimum length for abbreviations of synonyms or translations, count double byte characters and EBCDIC characters as single logical characters and ignore shift-out and shift-in characters.

For example, if you have the keyword 'abcd $\boxed{\text{SO}}$ k1k2k3 $\boxed{\text{SI}}$ efg', setting the minimum abbreviation of four allows 'abcd' as the shortest abbreviation. Setting the minimum abbreviation of five, would allow 'abcd $\boxed{\text{SO}}$ k1 $\boxed{\text{SI}}$ ' as the shortest abbreviation. Setting the minimum abbreviation of six allows 'abcd $\boxed{\text{SO}}$ k1k2 $\boxed{\text{SI}}$ ' as the shortest abbreviation, and so on.

- If you use DBCS characters when adding translations and translation synonyms to a DLCS file, you can issue CONVERT COMMANDS and SET LANGUAGE on these translations. However, you can only use these commands if the language you are using is set up as a double-byte language.

From CMS

- DBCS or mixed DBCS command names and keywords are accepted. DBCS strings cannot be specified for operand and option values such as filename, filetype, filemode, cuu, and so on.
- Each token in the tokenized PLIST is resolved to be a complete DBCS string. In other words, one of these tokens can contain no more than three double byte characters.
- When you invoke CMS commands, you can use DBCS EBCDIC to specify CMS delimiters such as blanks or parentheses.

Examples: Using the Parsing Facility

Creating a DLCS File

You have two commands, MYCMD1 and YOURCMD.

MYCMD1 has the following syntax:

```
MYCmd1 fn ft [ ( [DIsk|PRint] [NUMrecs nnn] [)] ]
```

YOURCMD has the following syntax:

```
YOURcmd string [ (TYPE [)] ]
```

Instead of coding syntax checking into your program, you plan to invoke the parsing facility for these commands. So you have to create a DLCS file to contain both syntax definitions.

You can create a CMS file called TEST DLCS to contain the statements for these commands that could look like this:

```
 1     :DLCS DMS USER AMENG :;
 2     :* The first command
 3       :CMD MMYCMD1 MYCMD1 MYCMD1 3 :;
 4         :SYN MY1 3 :;
 5         :OPR FCN(FN) :;
 6         :OPR FCN(FT) :;
 7         :OPT KWL(<DISK 2> <PRINT 2>) :;
 8         :OPT KWL(<NUMRECS 3>) FCN(PINTEGER) :;
 9     :* The second command
10       :CMD YYOURCMD YOURCMD YOURCMD 4 :;
11         :OPR FCN(STRING) :;
12         :OPT KWL(<TYPE 4>) :;
```

where:

Line Number	**Explanation**
1	Defines this file for the DMS application, the commands as user commands, and the ID of the language as AMENG.
2	A comment indicating the start of the first command syntax definition.
3	Defines MMYCMD1 as the unique-id for this syntax definition, and MYCMD1 as the command name with a minimum abbreviation of MYC.
4	Defines a synonym, MY1, for the command name with no abbreviation.
5	Specifies the first required operand is a filename.
6	Specifies the second required operand is a filetype.
7	Specifies two options: DISK as an option with a minimum abbreviation of DI, and PRINT as an option with a minimum abbreviation of PR.
8	Specifies another option as a keyword-value pair: NUMRECS as an option with a minimum abbreviation of NUM.
9	A comment indicating the start of the second command syntax definition.
10	Defines the unique-id and command name for this command definition.
11	Defines the only operand of this command as string.
12	Defines TYPE as the option with no abbreviation.

Creating a DLCS File with National Language Translations

You could also create TESTFRAN DLCS to contain national language translations for these two commands. If you wanted to include French translations, your file might look like this:

```
 1     :DLCS DMS USER FRANC :;
 2     :* The first command
 3       :CMD MMYCMD1 MYCMD1 FRANCMD1 8 :;
 4         :SYN MY1 3 :;
 5         :OPR FCN(FN) :;
 6         :OPR FCN(FT) :;
 7         :OPT KWL(<DISK 2 DISQUE 4> <PRINT 2 IMPRIMER 4>) :;
 8         :OPT KWL(<NUMRECS 3 NOMENREG 6>) FCN(PINTEGER) :;
 9     :* The second command
10       :CMD YYOURCMD YOURCMD VOTRECOM 5 :;
11         :OPR FCN(STRING) :;
12         :OPT KWL(<TYPE 4 AFFICHER 3>) :;
```

where:

Line Number	**Explanation**
1	Defines this file for the DMS application, the commands as user commands, and the ID of the language to be FRANC.
2	A comment indicating the start of the first command syntax definition.
3	Defines MMYCMD1 as the unique-id for this syntax definition, MYCMD1 as the command name, and FRANCMD1 as the national language name with no abbreviation.
4	Defines a synonym, MY1, for the command name with no abbreviation.
5	Specifies the first required operand is a filename.
6	Specifies the second required operand is a filetype.
7	Specifies two options: DISK as an option with a minimum abbreviation of DI, and DISQUE as the national language name with a minimum abbreviation of DISQ. PRINT as an option with a minimum abbreviation of PR, and IMPRIMER as the national language name with a minimum abbreviation of IMPR.
8	Specifies another option as a keyword-value pair: NUMRECS as an option with a minimum abbreviation of NUM, and NOMENREG as the national language name with a minimum abbreviation of NOMENR.
9	A comment indicating the start of the second command syntax definition.
10	Defines the unique-id, command name, and national language name for this command definition.
11	Defines the only operand of this command as string.
12	Defines TYPE as an option with no abbreviation, and AFFICHER as the national language name with a minimum abbreviation of AFF.

Calling the Parsing Facility from a REXX Program

You have a CMS file called TEST DLCS containing DLCS statements for the command MYCMD1 with the following syntax:

```
MYCmd1 fn ft [ ( [DIsk|PRint] [NUMrecs nnn] [)] ]
```

The syntax definition for this command is in the file TEST DLCS. See the example "Creating a DLCS File" on page 131 for the contents of TEST DLCS.

Once you use CONVERT COMMANDS to convert your file into a format that can be read in internally and use the SET LANGUAGE command to activate the language, you can invoke the parsing facility from an EXEC.

The following two sample REXX programs process MYCMD1. The first one illustrates a call to the parsing facility. The second does not call the parsing facility. The EXEC performs syntax checking.

```
/* This EXEC processes the MYCMD1 command with a format      */
/* as follows: MYCmd1 fn ft ( DIsk|PRint NUMrecs nnn )        */
/* The options may be omitted; the file name and type        */
/* cannot.                                                    */
address command

/* First, call the parser to check syntax of the command     */
/* string.                                                    */
'PARSECMD MMYCMD1'

If rc ¬= 0 then signal error    /* Go to ERROR if bad         */
                                /* string.                    */
/* The command string is valid, so we can search through     */
/* the tokens to find out what options were specified.       */
/* It does not matter what language is active, because      */
/* the parser has translated the command name and any       */
/* options that were given.                                   */
/*                                                            */
/* We know that:                                              */
/*                token.1= the command name MYCMD1;           */
/*                token.2= the passed file name;              */
/*                token.3= the passed file type;              */
/* if it exists,token.4=OPTSTART;                             */
/* and if they exist, remaining tokens -.5, .6, .7 -         */
/* could be TYPE, DISK, NUMRECS, or nnn.                      */

how_to_output = 'DISK'          /* Set default output to      */
                                /* disk.                      */
number = '*'                    /* Set number to the          */
                                /* whole file.                */
do i = 4 to token.0             /* Loop thru tokens, set      */
                                /* flags.                     */
  select
    when token.i = 'DISK' then how_to_output = 'DISK'
    when token.i = 'PRINT' then how_to_output = 'PRINT'
    otherwise                   /* Must be NUMRECS            */
      i = i + 1                 /* parameter.                 */
      number = token.i
  end
end

/* At this point, all of our flags and values have been      */
/* set, and we are ready to process the file.                */

. . .                                           */
```

Figure 14. Sample REXX Program 1

```
/* This EXEC processes the MYCMD1 command with a format       */
/* as follows: MYCmd1 fn ft ( DIsk|PRint NUMrecs nnn )        */
/* The options may be omitted; the file name and type         */
/* cannot.                                                     */
address command

/* First lets see what was passed to us                        */
arg fn ft '(' options
if fn='' | ft = '' then signal NAME_MISS /*parms missing?     */
'ESTATE' fn ft '*'          /* check validity of fn/ft        */
if rc = 20 then signal NAME_ERROR /* invalid fn or ft         */

how_to_output = ''                /* initialize to nulls      */
number = ''                       /* same here                */

do while options ¬= ''          /* loop thru all options       */
  parse var options opt options
  select;
    when (opt='DISK')             /* DISK specified           */
      then do              /* ensure DISK/TYPE only once      */
             if how_to_output¬='' then signal HOW_ERROR
             how_to_output = 'DISK'
           end
    when (opt='TYPE')             /* TYPE specified           */
      then do              /* ensure DISK/TYPE only once      */
             if how_to_output¬='' then signal HOW_ERROR
             how_to_output = 'TYPE'
           end
    when (opt='NUM'|opt='NUMR'|opt='NUMRE'|opt='NUMREC'|opt='NUMRECS')
      then do              /* verify validity of number       */
             if number¬='' then signal NUM_ERROR
             parse var options num options
             if num = '' then signal NO_NUM_ERROR
             if ¬datatype(num,w) then signal INV_NUM_ERROR
             if num <= 0 then signal INV_NUM_ERROR
             number = num
           end
    otherwise                     /* unknown option           */
      signal INVALID_OPTION
  end
end
```

Figure 15 (Part 1 of 2). Sample REXX Program 2

```
/* We must set the defaults.                               */
if how_to_output = '' then how_to_output = 'DISK'
if number = '' then number = '*'

signal OK
 /* Error Routines                                       */

 NAME_MISS:
   say 'File name and file type must be specified'
   signal EXIT

 NAME_ERROR:
   say '"'fn ft'" is an invalid file ID'
   signal EXIT

 HOW_ERROR:
   say how_to_output 'already specified, "'opt'" is invalid'
   signal EXIT

 NUM_ERROR:
   say 'NUMRECS specified twice'
   signal EXIT

 NO_NUM_ERROR:
   say 'The value for NUMRECS has been omitted'
   signal EXIT

 INV_NUM_ERROR:
   say '"'num'" is an invalid positive number for NUMRECS'
   signal EXIT

 INVALID_OPTION:
   say '"'opt'" is an invalid option'
   signal EXIT

 OK:
 /* At this point, all of our flags and values have been  */
 /* set, and we are finally ready to process the file.
 . . .                                 */
```

Figure 15 (Part 2 of 2). Sample REXX Program 2

Calling the Parsing Facility from an Assembler Program

The following two sample programs perform the same task. The first program invokes the parsing facility via PARSECMD. In this way, you do not need extra code to handle parsing. The second program includes code for parsing abbreviations, missing operands, and extra operands as well as code that issues error messages.

```
**********************************************************************
*
* ROUTINE:  SSORT
* FUNCTION: TAKE 2 STRINGS AND DISPLAY THEM IN EITHER ASCENDING
*           OR DESCENDING ORDER
* SYNTAX:   SSORT {ASCENDING|DESCENDING} STRING1 STRING2
* DLCS:     :CMD FFSSORT SSORT SSORT 5 :;
*             :OPR KWL(<ASCENDING 3><DESCENDING 4>) :;
*             :OPR FCN(STRING) :;
*             :OPR FCN(STRING) :;
*
**********************************************************************
SSORT    START
         USING *,12
         LR    12,15                ESTABLISH ADDRESSABLILITY
         ST    14,R14SAVE           SAVE RETURN ADDRESS
**********************************************************************
* PARSE SSORT COMMAND
**********************************************************************
         LA    3,PARSLBL            GET ADDRESS OF PARSERCB STORAGE
         USING PARSERCB,3
         PARSECMD MF=(E,PARSLBL),UNIQID=UID,PLIST=(1),                *
               EPLIST=(0),ERROR=EXIT
         USING PVCENTRY,10
         L     10,PARPVCAD          GET PARSER VALIDATION CODE TABLE
         L     10,PVCNEXTA          POINT TO ENTRY OF ASCEND/DESCEND OPR
         L     9,PVCETOKA           GET ADDRESS OF ASCEND/DESCEND OPR
         L     10,PVCNEXTA          POINT TO ENTRY OF 1ST STRING
         L     5,PVCETOKA           GET ADDRESS OF 1ST STRING
         L     6,PVCETOKL           GET LENGTH OF 1ST STRING
         L     10,PVCNEXTA          POINT TO ENTRY OF 2ND STRING
         L     7,PVCETOKA           GET ADDRESS OF 2ND STRING
         L     8,PVCETOKL           GET LENGTH OF 2ND STRING
**********************************************************************
* DISPLAY STRING1 AND STRING2 IN EITHER ASCENDING OR DESCENDING ORDER
**********************************************************************
         CR    6,8                  WHICH STRING HAS FEWER CHARS ?
         BH    COMP2                2ND STRING, TAKE BRANCH
         BCTR  6,0                  DECREMENT FOR EXECUTE
         EX    6,COMPARGS           COMPARE STRINGS
         LA    6,1(,6)              INCREMENT BACK
         BNH   SMALL1               IF 1ST STRING GOES 1ST, BRANCH
         B     SMALL2               IF 2ND STRING GOES 1ST, BRANCH
```

Figure 16 (Part 1 of 2). Sample Assembler Program 1

```
COMP2    DS    0H
         BCTR  8,0                DECREMENT FOR EXECUTE
         EX    8,COMPARGS         COMPARE STRINGS
         LA    8,1(,8)            INCREMENT BACK
         BNL   SMALL2             IF 2ND STRING GOES 1ST, BRANCH
SMALL1   DS    0H
         CLI   0(9),C'D'          WANT TO SORT IN DESCENDING ORDER ?
         BE    TYPE21             YES, TYPE 2ND FOLLOWED BY 1ST
TYPE12   DS    0H
         WRTERM (5),(6)           WRITE OUT THE 1ST STRING
         WRTERM (7),(8)           WRITE OUT THE 2ND STRING
         B     GOODEXIT           EXIT WITH RC = 0
SMALL2   DS    0H
         CLI   0(9),C'D'          WANT TO SORT IN DESCENDING ORDER ?
         BE    TYPE12             YES, TYPE 1ST FOLLOWED BY 2ND
TYPE21   DS    0H
         WRTERM (7),(8)           WRITE OUT THE 2ND STRING
         WRTERM (5),(6)           WRITE OUT THE 1ST STRING
GOODEXIT DS    0H
         SR    15,15              ZERO OUT RC
EXIT     DS    0H
         L     14,R14SAVE         GET RETURN ADDRESS
         BR    14                 RETURN
PARSLBL  PARSECMD MF=L            GET INITIALIZED PARSERCB
UID      DC    CL16'FFSSORT'      UNIQUE ID FOR PARSECMD
R14SAVE  DS    A                  RETURN ADDRESS
COMPARGS CLC   0(*-*,5),0(7)      COMPARE STRINGS
         PARSERCB
         PVCENTRY
         END
```

Figure 16 (Part 2 of 2). Sample Assembler Program 1

```
**********************************************************************
*
* ROUTINE:  LSORT
* FUNCTION: TAKE 2 STRINGS AND DISPLAY THEM IN EITHER ASCENDING
*           OR DESCENDING ORDER
* SYNTAX:   LSORT {ASCENDING|DESCENDING} STRING1 STRING2
* REQUIREMENTS: MUST GENMOD WITH SYSTEM OPTION
*
**********************************************************************
LSORT    START
         USING *,12
         LR    12,15                ESTABLISH ADDRESSABLILITY
         ST    14,R14SAVE           SAVE RETURN ADDRESS
**********************************************************************
* PARSE LSORT COMMAND
**********************************************************************
         LR    11,0                 GET EPLIST ADDRESS
         USING EPLIST,11
         L     9,EPLARGBG           GET ADDRESS OF 1ST ARG
         L     10,EPLARGND          GET END OF ARGS ADDRESS
         DROP  11
         SR    10,9                 GET LENGTH OF ARGS FIELD
         LTR   10,10                DOES ARG1 EXIST ?
         BZ    MISSARG1             NO, ISSUE MESSAGE
         LA    1,0(10,9)            POINT PAST END OF ARGS FIELD
         LR    3,9                  GET ADDRESS FOR EXECUTE
         BCTR  10,0                 DECREMENT FOR EXECUTE
         EX    10,FINDEND           FIND END OF ARG 1
         LA    10,1(,10)            INCREMENT BACK
         SR    1,9                  GET LENGTH OF ARG 1
         BZ    MISSARG1             IF LENGTH 0, MISSING ARG 1
         LR    11,1                 SAVE LENGTH OF ARG 1
         BCTR  11,0                 DECREMENT FOR EXECUTE
         EX    11,UPCASE            UPPERCASE ARG 1
         LA    11,1(,11)            INCREMENT BACK
TRYASC   DS    0H
         C     11,ASCMINL           ARG LENGTH LESS THAN MIN FOR ASCEND ?
         BL    TRYDESC              YES, TRY DESCENDING
         C     11,ASCMAXL           ARG LENGTH TOO BIG FOR ASCENDING ?
         BH    TRYDESC              YES, TRY DESCENDING
         BCTR  11,0                 DECREMENT FOR EXECUTE
         EX    11,COMPASC           SEE IF ASCENDING WAS SPECIFIED
         LA    11,1(11)             INCREMENT BACK
         BE    GETSTRG1             IF ASCENDING, GET STRINGS
```

Figure 17 (Part 1 of 4). Sample Assembler Program 2

```
TRYDESC  DS    OH
         C     11,DESCMINL          ARG LENGTH LESS THAN MIN FOR DESCEND?
         BL    BADARG1              YES, ARG1 IS BAD
         C     11,DESCMAXL          ARG LENGTH TOO BIG FOR DESCENDING ?
         BH    BADARG1              YES, ARG1 IS BAD
         BCTR  11,0                 DECREMENT FOR EXECUTE
         EX    11,COMPDESC          SEE IF ASCENDING WAS SPECIFIED
         LA    11,1(11)             INCREMENT BACK
         BNE   BADARG1              IF NOT DESCENDING, ARG IS BAD
GETSTRG1 DS    OH
         SR    10,11                ADJUST LENGTH OF ARGS FIELD
         BZ    MISSARG2             IF 0, MISSING ARG 2
         LA    4,0(11,9)            POINT PAST END OF ARG 1
         LR    1,4                  GET FOR EXECUTE
         BCTR  10,0                 DECREMENT FOR EXECUTE
         EX    10,FINDSTRT          FIND START OF STRING 1
         LA    10,1(,10)            INCREMENT BACK
         BZ    MISSARG2             ARG 2 MISSING, ISSUE MESSAGE
         LR    5,1                  REMEMBER ADDRESS OF STRING1
         LR    2,5                  GET ADDRESS OF STRING 1
         SR    2,4                  - ADDRESS OF 1ST DEL AFTER ARG 1
         SR    10,2                 ADJUST LENGTH OF ARGS FIELD
         LA    1,0(10,5)            POINT PAST END OF ARGS FIELD
         LR    3,5                  GET ADDRESS OF STRING 1
         BCTR  10,0                 DECREMENT FOR EXECUTE
         EX    10,FINDEND           FIND END OF STRING 1
         LA    10,1(,10)            INCREMENT BACK
         BZ    MISSARG3             NO DELIMS, MISSING STRING 2
         SR    1,5                  GET LENGTH OF STRING 1
         BZ    MISSARG2             IF LENGTH 0, MISSING ARG 2
         LR    6,1                  SAVE LENGTH OF STRING 1
GETSTRG2 DS    OH
         SR    10,6                 ADJUST LENGTH OF ARGS FIELD
         BZ    MISSARG3             IF 0, MISSING ARG 3
         LA    4,0(6,5)             POINT PAST END OF STRING 1
         LR    1,4                  GET FOR EXECUTE
         BCTR  10,0                 DECREMENT FOR EXECUTE
         EX    10,FINDSTRT          FIND START OF STRING 2
         LA    10,1(,10)            INCREMENT BACK
         BZ    MISSARG3             ARG 3 MISSING, ISSUE MESSAGE
         LR    7,1                  REMEMBER ADDRESS OF STRING 2
         LR    2,7                  GET ADDRESS OF STRING 2
         SR    2,4                  - ADDRESS OF 1ST DEL AFTER ARG 2
         SR    10,2                 ADJUST LENGTH OF ARGS FIELD
         LA    1,0(10,5)            POINT PAST END OF ARGS FIELD
         LR    3,7                  GET ADDRESS OF STRING 2
         BCTR  10,0                 DECREMENT FOR EXECUTE
```

Figure 17 (Part 2 of 4). Sample Assembler Program 2

```
         EX    10,FINDEND          FIND END OF STRING 2
         LA    10,1(,10)           INCREMENT BACK
         BZ    GETLEN              NO DELIMS, USE LENGTH OF ARGS FIELD
         SR    1,7                 GET LENGTH OF STRING 2
         BZ    MISSARG3            IF LENGTH 0, MISSING ARG 3
         LR    8,1                 SAVE LENGTH OF STRING 2
         B     CHKEXTRA            SEE IF EXTRA OPERANDS SPECIFIED
GETLEN   DS    0H
         LR    8,10                USE LENGTH LEFT OF ARGS FIELD
CHKEXTRA DS    0H
         SR    10,8                ADJUST LENGTH OF ARGS FIELD
         BZ    SORTSTRG            IF 0, ALL OK
         LA    4,0(8,7)            POINT PAST END OF STRING 2
         LR    1,4                 GET FOR EXECUTE
         BCTR  10,0                DECREMENT FOR EXECUTE
         EX    10,FINDSTRT         FIND EXTRA OPERANDS
         LA    10,1(,10)           INCREMENT BACK
         BZ    SORTSTRG            NO EXTRA OPERANDS, ALL OK
         LR    2,1                 GET ADDRESS OF EXTRA OPERANDS
         SR    2,4                 - ADDRESS OF 1ST DEL AFTER ARG 3
         SR    10,2                GET LENGTH OF EXTRA OPERANDS
         LR    2,1                 GET ADDRESS OF EXTRA OPERANDS
         B     EXTRAOP             EXTRA OPERANDS, ISSUE MESSAGE
**********************************************************************
* DISPLAY STRING1 AND STRING2 IN EITHER ASCENDING OR DESCENDING ORDER
**********************************************************************
SORTSTRG DS    0H
         CR    6,8                 WHICH STRING HAS FEWER CHARS ?
         BH    COMP2               2ND STRING, TAKE BRANCH
         BCTR  6,0                 DECREMENT FOR EXECUTE
         EX    6,COMPARGS          COMPARE STRINGS
         LA    6,1(,6)             INCREMENT BACK
         BNH   SMALL1              IF 1ST STRING GOES 1ST, BRANCH
         B     SMALL2              IF 2ND STRING GOES 1ST, BRANCH
COMP2    DS    0H
         BCTR  8,0                 DECREMENT FOR EXECUTE
         EX    8,COMPARGS          COMPARE STRINGS
         LA    8,1(,8)             INCREMENT BACK
         BNL   SMALL2              IF 2ND STRING GOES 1ST, BRANCH
SMALL1   DS    0H
         CLI   0(9),C'D'           WANT TO SORT IN DESCENDING ORDER ?
         BE    TYPE21              YES, TYPE 2ND FOLLOWED BY 1ST
TYPE12   DS    0H
         WRTERM (5),(6)            WRITE OUT THE 1ST STRING
         WRTERM (7),(8)            WRITE OUT THE 2ND STRING
         B     GOODEXIT            EXIT WITH RC = 0
```

Figure 17 (Part 3 of 4). Sample Assembler Program 2

```
SMALL2   DS    0H
         CLI   0(9),C'D'          WANT TO SORT IN DESCENDING ORDER ?
         BE    TYPE12             YES, TYPE 1ST FOLLOWED BY 2ND
TYPE21   DS    0H
         WRTERM (7),(8)           WRITE OUT THE 2ND STRING
         WRTERM (5),(6)           WRITE OUT THE 1ST STRING
GOODEXIT DS    0H
         SR    15,15              ZERO OUT RC
         B     EXIT               EXIT
EXTRAOP  DS    0H
         APPLMSG NUM=070,CSECT=SSO,SUB=(CHARA,((2),(10)))
         B     BADRC              EXIT WITH RC = 24
BADARG1  DS    0H
         APPLMSG NUM=388,CSECT=SSO,SUB=(CHARA,((9),(11)))
         B     BADRC              EXIT WITH RC = 24
MISSARG1 DS    0H
MISSARG2 DS    0H
MISSARG3 DS    0H
         APPLMSG NUM=386,CSECT=SSO
BADRC    DS    0H
         LA    15,24              GET BAD RC
EXIT     DS    0H
         L     14,R14SAVE         GET RETURN ADDRESS
         BR    14                 RETURN
FINDEND  TRT   0(*-*,3),DELIMS    FIND NEXT DELIMITER
FINDSTRT TRT   0(*-*,4),NONDELIM  FIND NEXT NON-DELIMITER
UPCASE   OC    0(*-*,9),BLANKS    UPPERCASE ARGUMENT
COMPASC  CLC   0(*-*,9),ASCEND    CHECK FOR ASCENDING OPERAND
COMPDESC CLC   0(*-*,9),DESCEND   CHECK FOR DESCENDING OPERAND
COMPARGS CLC   0(*-*,5),0(7)      COMPARE STRINGS
         DS    0F
BLANKS   DC    10C' '             BLANKS FOR UPPERCASING OPERAND
DELIMS   DC    64X'00',C' ',12X'00',C'(',178X'00'
NONDELIM DC    64X'FF',X'00',12X'FF',X'FF',178X'FF'
ASCEND   DC    C'ASCENDING'
ASCMINL  DC    F'3'
ASCMAXL  DC    F'9'
DESCEND  DC    C'DESCENDING'
DESCMINL DC    F'4'
DESCMAXL DC    F'10'
R14SAVE  DS    A                  RETURN ADDRESS
         DSECT
EPLIST   DS    0H
EPLCMD   DS    A
EPLARGBG DS    A
EPLARGND DS    A
         END
```

Figure 17 (Part 4 of 4). Sample Assembler Program 2

Creating and Distributing Your Own CMS Commands

Using DLCS

If you give users commands that call the parsing facility, put the syntax in a DLCS file that has a unique application identifier. In this way, users who receive your commands and syntax do not have to merge the syntax definition with their DMS tables.

Refer to the example on page 132. In this example, you could change the application identifier on the DLCS statement to your initials. For example, if your initials are AGW, the DLCS statement looks like:

```
:DLCS AGW USER AMENG :;
```

and the call to PARSECMD in REXX is:

```
'PARSECMD MMYCMD1 (APPLID AGW'
```

To make it even easier for other users, you can automatically load and drop the table from storage by inserting

```
'SET LANGUAGE (ADD AGW USER'
```

in your REXX program just before the call to PARSECMD, and

```
'SET LANGUAGE (DELETE AGW USER'
```

just before exiting. By doing this, all parsing is hidden, and users do not have to issue the SET LANGUAGE command.

Defining Translations, Synonyms and Abbreviations

If users want to translate the command or the keywords of the command into their own national language, they have to edit the DLCS table you send them to translate the parameters. However, to translate commands or keywords into a language other than the default language, the other language must exist on your system. See "Chapter 16. Getting National Languages on Your System" in the *VM System Facilities for Programming* for more details.

If users want to define a translation for the command name, they can just add an entry in their DMS table. For example, if the user's translation for your command name is 'FRIENDCMD', the DMS table entry is

```
:CMD ' ' MYCMD1 FRIENDCMD 5 :;
```

If users just want to abbreviate your command name, they can add an entry in their own DMS user tables that defines your command with the blank uniqueid:

```
:CMD ' ' MYCMD1 MYCMD1 3:;
```

To abbreviate the command name and define a synonym, such as 'MC', they can add:

```
:CMD ' ' MYCMD1 MYCMD1 3:;
  :SYN MC 2:;
```

After users define translations, synonyms and abbreviations, they must run CONVERT COMMANDS against the files they have changed.

Note: This application does not support DBCS tokens unless there is already a system table available for the application.

Defining HELP Files

You can also create HELP text files for your own commands. The HELP files contain information about these commands. By specifying the appropriate HELP command, you can display information about the commands you created. See the *VM/SP CMS User's Guide* for details on creating your own HELP files.

Using Message Repository Files

When you write a program and you want error messages to be displayed, you can put message text directly in your program. For example, in Assembler programs you can use the LINEDIT or other macros to display messages on the screen; in REXX programs, you can just use a SAY statement to display whatever message you want.

However, if you have many messages, your programs can become cluttered. Instead of coding message text directly in a program, you can store all your message texts in one file called a "repository." Then, when you want to display a message, you can retrieve the message text you want from this repository.

Having all message text in one central file has the following advantages:

1. Message text will not clutter your program.

2. You can access the same message from many programs without having to specify the message text each time.

3. If you would like your messages to be available in a language other than English, translation centers can just translate your single message file. You can then have your messages in the language you want.

You can create your own message file for whatever application you want to run, including CMS.

For CMS system messages, a source repository file is already built for you; it has a fileid "DMSMES REPOS". You can XEDIT this file to view messages. You can also print off a copy of the CMS message file so you can refer to it when you want to call a CMS message from your program.

Note: *For languages other than English available on your system, the filename of the CMS message repository is different. See "Chapter 16. Getting National Languages on Your System" in VM System Facilities for Programming for more details.*

Overview of Creating and Using a Message Repository

Five steps are involved with creating and then using your own message repository file. These steps are:

1. Creating a message repository file using XEDIT.
2. Checking the accuracy of the message repository notation.
3. Converting the repository into machine readable form.
4. Making the repository file available for the language you are working with.
5. Accessing the messages from programs using the APPLMSG macro or the XMITMSG command.

Creating a Message File: When you create a message repository file, you must follow certain notation rules. These rules are explained in "Rules for Making Your Own Repository" on page 148.

Checking the Accuracy of the Message File: After you create a message repository, you should check to see if you made any incorrect entries. You can use the GENMSG command with the NOOBJECT option to flag any invalid syntax statements:

```
GENMSG fn ft fm applid (NOOBJECT
```

(`fn ft fm` is the fileid of your message repository you created in XEDIT. `applid` is the identifier for your application. It must be three characters. For example, if you made the message file for an accounting application, you might want to have your applid be "ACT".

Note: Be sure to record the applid you choose. You will need to reference it when you access your messages, and you will also need the applid if you create command syntax files.

GENMSG displays an error message in the listing file for any invalid statement it finds in the message file. If you have any errors, just XEDIT the file and correct any mistakes.

Compiling the Message File: When your message repository contains no more syntax errors, you then have to "compile" your message repository file. This step converts your file into an machine readable form that the message processor recognizes. The GENMSG command also does this task:

```
GENMSG fn ft fm applid
```

GENMSG creates an output file that contains the internal version of your message repository. The filename of this output file is the same as the input file; the filetype is "TXT*langid*," where *langid* is the language identifier. The default value for *langid* is the language currently in use.

See the *VM/SP CMS Command Reference* for details on the GENMSG command.

Making the Message File Available: Once the message file has been compiled, you must make the message file active for the language you are working in. You accomplish this with the SET LANGUAGE command.

For example, suppose you created and compiled a message repository file that is an addition to the CMS system repository. Before issuing the SET LANGUAGE command, your compiled message file must have a filename of DMSUME and a filetype of TXT*langid.* To get your message file active, you could issue the SET LANGUAGE command as follows:

```
SET LANGUAGE (ADD DMS USER
```

See the *VM/SP CMS Command Reference* for a complete description of the SET LANGUAGE command.

Accessing the Messages: You can access messages from a repository in two different ways:

1. From EXEC 2 EXECs, REXX programs, or CMS, use the XMITMSG command.

   For example, to display this CMS message from a REXX program:

   ```
   File MSG TEST not found
   ```

   you could use the following XMITMSG call:

   ```
   "XMITMSG 002 'MSG' 'TEST' (DISP FOR 1 APPLID DMS COMP"
   ```

   (See the *VM/SP CMS Command Reference* for a complete description of XMITMSG.)

2. From assembler programs, use the APPLMSG macro.

   For example, to display this CMS message from an assembler program:

   ```
   File MSG TEST not found
   ```

   you could use the following APPLMSG call (assuming the address of MSG is in register 5 and the address of TEST is in register 6):

```
APPLMSG   NUM=002,FMT=1,
            APPLID=DMS,COMP=YES,SUB=(CHARA,((5),3)),
            CHARA,((6),4)),DISP=TYPE,TYPCALL=SVC
```

(See the *VM/SP CMS Macros and Functions Reference* for a complete description of APPLMSG.)

Rules for Making Your Own Repository

Each message record in a repository file contains five fields. When you create your file using XEDIT, you must make every message record in the following format:

```
          NNNNFFLLS ------------------------ text ------------------------------
columns   1   5 7 9 11                                                        72
```

where:

NNNN
is the message number, in columns 1-4. You can use a 4 digit message number in a message repository, but by default only the last three digits are displayed in the message header. You can use the first digit to group messages; for example, the first digit of all dictionary records (see "Dictionary Substitution" on page 151) could be '8'.

You do not have to place messages in sequence by message number in a CMS (or CMS application) user repository. However, message numbers do have to be in sequential order in a CP repository.

FF
is the message format, in columns 5-6. This field is for a message that can be in several versions. If a message has just one format, you do not need to type anything -- the format field defaults to "01". You cannot use "00" as a format number.

LL
is the line number of the message, in columns 7-8. This field is used if you want the text for a single message format to be displayed on more than one line. Messages that spread across more than one line must have sequential, consecutive line numbers.

If a single format of a message has just one line, you do not need to type a line number -- the line number defaults to "01". You cannot use "00" as a line number.

S
is the severity code, in column 9. Your severity codes should be one of the following:

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

text
is the message text, starting in column 11. You can specify up to 62 characters of message text on one line. If the text for a single message is longer than 62 characters, you must put the message text on more than one line and specify a line number for each.

If you want multi-line message text displayed on the screen in one continuous line (wrapped around), the message number (NNNN), the message format (FF), and the line number (LL) must be identical for each line.

If you want multi-line message text displayed on the screen in more than one line, you must make the first line number 01 and line numbers after that 02, 03, etc.

Your message repository file should also contain comment records. These must start with an asterisk in column 1.

```
* This is an example of a comment line
```

Comment lines, which can go anywhere in a message repository file, should describe what is in the file.

Finally, you must put a control line in your external repository. This control line specifies two things:

1. A character that specifies substitutions. This must be the first non-blank character on the control line.
2. A number that specifies the amount of message number digits (3 or 4) you want to display. This must be the second non-blank character on the control line.

The control line must be the first non-comment record.

Example:

The following example shows an external repository file, which is stored on a disk. You can view, edit, and update this message repository. (Translation centers can also translate this to another language.)

```
*
* This is an example of a message repository file for a small
* programming application.
* This was created via XEDIT.
* You can code a file similar to this for your own application.
*
& 3 SPECIFIES THE SUBSTITUTION CHARACTER + NO. OF DIGITS
0005    E Invalid syntax; please reissue command.
0015    R Enter the number of copies you want:
002501  I Function has completed
002502  I Subroutine has completed
0100  01R Your program has just halted at label ABCD.
0100  02  You can quit the program by entering 'Q', or
0100  03  press the ENTER key to continue
```

Here is a line by line description of what this repository contains:

Line Number(s)	**Explanation**
1 -- 6	Comment lines.
7	The control line. The first non-blank character on this line specifies the substitution character for messages -- &. (See "Substitution in Messages" on page 151) The second non-blank character specifies that you want to display only three message number digits (the default). *Note: If a message number is greater than 999, then 4 digits are displayed regardless of the control line number.*
8	The first message in the repository, number 0005. Message 0005 has only one format and takes up only one line in the file; so the FORMAT columns (5-6) and the LINE columns (7-8) are blank. The message results from a user error, so the severity (column 9) is "E".
9	The second message, number 0015. This message has only one format and takes up only one line; so the FORMAT columns (5-6) and the LINE columns (7-8) are blank. The message is requesting input from a user, so the severity (column 9) is "R".
10 -- 11	The third message, number 0025. This message number has two formats; depending on the error, either `Function has completed` (format 01) or `Subroutine has completed` (format 02) is displayed. These messages just give the user information, so their severity is "I."

12 -- 14 The fourth and final message, number 0100. This message has only one format, but it spreads across three lines of the repository. Columns (7-8) show the line numbers of this message. The message requests input from a user, so the severity is "R."

Substitution in Messages

In the above example, the text for each message is the same every time the message is displayed. However, you will probably want to have some message texts that are similar, but say different things depending on the situation. For example, you might have a message that says:

```
Invalid option 'GO'
```

But you also want to have these messages in your repository:

```
Invalid option 'FILE'
Invalid option 'RUN'
Invalid option 'STOP'
```

You do not need four separate messages in your repository. Instead, you can have a single message text, and then substitute different information for the bad option. The single message looks like this:

```
Invalid option '&1'
```

Messages that require substitutions have parameters in a form defined by the user (eg. &1, &2 ...). These parameters show the placement of the substitutions and their order. The first character in the first non-commentary record of the external repository defines the substitution character. This character may not be a DBCS character.

Here are some rules about substitutions:

- A substitution may be a single word, a phrase, or an entire sentence.
- A substitution can go anywhere within a message.
- You can have more than one substitution per message.

The data that replaces the &1, &2, etc. can come from the program itself (a parameter on the APPLMSG or XMITMSG call) or from a dictionary.

Dictionary Substitution

Each dictionary record contains a 4 digit message identifier and dictionary text. These records are stored in the repository file along with the messages. You can make the first digit of the message identifier a certain number (8, for example) that shows the item is a dictionary item.

Example:

Here is an example of a message repository that contains a two-item dictionary:

```
*
* This is an example of a message repository file made via XEDIT
* You can code a file similar to this for your application.
*
& 4 LINE SPECIFIES THE SUBSTITUTION CHARACTER + NO. OF DIGITS
0007    E Invalid option '&1'
0017    I You have invoked the &1
8001      compiler
8002      assembler
```

Here is a line by line description of what this repository contains:

Line Number(s)	Explanation
1 -- 4	Comment lines.
5	The control line. An ampersand (&) is the substitution character, and you want to display 4 message number digits.
6	The first message in the repository, number 0007. When the message is to be displayed, you have to specify what information is to be substituted in place of the &1.
7	The second message, number 0017. When the message is to be displayed, you have to specify what information is to be substituted in place of the &1.
8	A dictionary item, number 8001. If you want to call message 0017 and specify `DICT=8001` on a APPLMSG call (or just 8001 on a XMITMSG call), the following message is displayed: `aaammm0017I  You have invoked the compiler` (The message header also includes the application id aaa and the program name within the application mmm.)
9	A second dictionary item, numbered 8002. If you want to call message 0017 and specify `DICT=8002` on a APPLMSG call (or just 8002 on a XMITMSG call), the following message is displayed: `aaammm0017I  You have invoked the assembler`

When you code the call to access a message (via APPLMSG or XMITMSG), you just have to specify the dictionary number.

Creating Your Own CMS Messages

The CMS system repository uses three or four digits for CMS messages. It also uses 8000-8nnn for dictionary items and unnumbered responses. You can view this file using XEDIT, and you should print off a copy for you to use as reference.

You can create your own CMS messages and put them in a repository. Once you compile the message file (using GENMSG) and make the file active (using SET LANGUAGE), you can access your own messages.

A user CMS repository can contain message numbers that are additions to existing CMS messages or duplicate message numbers. If your CMS repository contains message numbers that duplicate existing CMS messages, your version overrides the system version.

For example, suppose you wanted to add your own informational message that says:

```
The command you issued takes five minutes to complete.
```

You can enter this message in your CMS repository in two different ways:

1. Using your own unique message number.

   You can look in the CMS system repository -- file "DMSMES REPOS" -- and find a number that is not currently being used for a CMS message. For example, if number 1000 is not currently being used, you can put this message in your repository as number 1000, compile the new repository file, and make the file active. You can then access the message using the command:

   ```
   XMITMSG 1000 (DISP FOR 1 APPLID DMS COMP
   ```

2. Using an existing CMS message number.

   The CMS system repository contains a message

   ```
   046E  No library name specified
   ```

   Suppose you enter the message

   ```
   The command you issued takes five minutes to complete.
   ```

   as number 046E in your repository, compile the repository file, and then make it active. When you access message 046E, you do not see the CMS system message, you see your own version of 046E.

If any of your own messages require dictionary substitutions, you should note this restriction: *You must include dictionary items for your messages in your own message repository—you cannot access a message from your own repository using dictionary items from the CMS message repository.*

Creating Your Own HELP Files

You can also create HELP text files for your own messages. The HELP files contain explanatory information about these messages. By specifying the appropriate HELP command, you can display information about the messages you created. See the *VM/SP CMS User's Guide* for details on creating your own HELP files.

Making Your Messages Available to Others

When you make your own repository and issue a SET LANGUAGE command, that repository is available only to your virtual machine. However, you may want to allow other users on your system to access your messages. You can accomplish this by either of the following two methods:

1. Have other users link to your disk. They must then issue a SET LANGUAGE command for their virtual machine.

2. Have your message file placed in shared storage so all users can access it. See "Chapter 16. Getting National Languages on Your System" in *VM System Facilities for Programming* for details.

Creating Immediate Commands

In addition to the CMS built-in immediate commands, CMS provides facilities for you to create your own immediate commands. Rules for creating your own immediate commands are as follows:

1. Immediate commands can be created in three ways:

    a. Immediate commands can be created from Assembler Language programs by issuing the IMMCMD macro. This macro associates a user-defined immediate command name with the address of a user-supplied exit routine that receives control when the immediate command is issued. Established immediate commands can also be explicitly cancelled by the IMMCMD macro. If not explicitly cancelled, all immediate commands created by the IMMCMD macro are automatically cancelled either upon return to the CMS command environment (if not in CMS SUBSET mode) or by entry to CMS abend.

    b. Immediate commands can be created from EXECs by use of the IMMCMD command. This command establishes and cancels immediate commands and determines the status of the immediate command. All immediate commands not explicitly cancelled by the IMMCMD command are automatically cancelled either upon return to the CMS command environment (if not in CMS SUBSET mode) or by entry to CMS abend. User exit routines cannot be used with immediate commands established by the IMMCMD command.

c. Immediate commands can be created by using the immediate attribute that is supported by the NUCEXT function and the NUCXLOAD command. When a nucleus extension is declared with the immediate attribute, that nucleus extension is established as an immediate command. By allowing nucleus extensions to be declared as immediate commands, the following additional flexibility is provided:

   1) Immediate command routines can be created in free storage.

   2) Immediate commands can be permanently established for the duration of a CMS IPL (that is, they are not cleared during CMS end-of-command processing).

   3) Immediate commands can be invoked as exits during abend (SERVICE attribute) and end-of-command (ENDCMD attribute) processing.

   4) Immediate commands can be established to survive CMS abend (SYSTEM attribute).

   Nucleus extensions established as immediate commands can be invoked as immediate commands or as part of normal SVC 202 processing. When a nucleus extension is called as an immediate command, the high-order byte of register 1 is set to X'06'.

   The immediate attribute is supported by the NUCEXT function (DECLARE, QUERY, CANCEL) and by the NUCXLOAD, NUCXDROP, and NUCXMAP commands.

2. Immediate commands can be 1 to 8 characters in length. Synonyms can be set up for immediate commands just like they can be for regular CMS commands. Immediate commands or their synonyms must begin with a non-blank character.

3. Immediate commands are delimited by a blank. Any data following the blank is passed to the immediate command routine as parameters. The capability to pass parameters is not applicable to immediate commands declared by the IMMCMD command. Immediate commands and their parameters are subject to translation just as regular CMS commands are.

4. Immediate commands can be set up to override built-in CMS Immediate commands (for example, HX). However, built-in CMS commands cannot be cleared.

5. Immediate commands with the same name can override each other in a stack-like manner, with the most recent one declared being the one in effect.

6. The logical line end character is ignored on immediate command input lines.

7. Both the IMMCMD macro, the NUCEXT function, and the NUCXLOAD command provide the capability to give control to an "exit" routine whenever a specific immediate command is invoked. These exit routines receive control as an extension of CMS I/O interrupt handing. Therefore, they receive control with a PSW key of 0 and are disabled for interrupts. The exit routine must not perform any I/O operations or issue any SVCs that result in I/O operations. In addition, the exit routine must not enable itself for interrupts. DIAGNOSE instructions can be used within the exit, but the exit routine must not enable itself for interruptions that may be caused by the DIAGNOSE (for example, DIAGNOSE code X'58').

8. Terminal users may optionally require that all immediate commands be prefixed with an escape character. Use the SET IMESCAPE command to set the escape character. The status of IMESCAPE function can be determined by the QUERY command. For more details, see the *VM/SP CMS Command Reference.*

Chapter 8. Developing OS Programs under CMS

CMS simulates many of the functions of the Operating System (OS), allowing you to compile, execute and debug OS programs interactively. For the most part, you do not need to be concerned with the CMS OS simulation routines; they are built into the CMS system. Before you can compile and execute OS programs in CMS, however, you must be acquainted with the following:

- Using OS data sets in CMS
- How to use the FILEDEF command
- Creating CMS files from OS data sets
- Using CMS libraries
- OS macros that CMS can simulate

These topics are discussed below. Additional information for OS VSAM users is in "Chapter 10. Using Access Method Services and VSAM under CMS and CMS/DOS" on page 273.

Note: The CMS system uses many OS terms, but there are a number of OS functions that CMS performs somewhat differently. Refer to Figure 18 on page 158 to help you become familiar with some of the equivalents (where they do exist) for OS terms and functions. It lists some commonly-used OS terms and discusses how CMS handles the functions they imply.

OS Term/Function	CMS Equivalent
catalogued procedure	EXEC files can execute command sequences similar to catalogued procedures, and provide for conditional execution based on return codes from previous steps.
data set	Data sets are called files in CMS. CMS can simulate certain OS data sets and can read real OS data sets only if they are sequential or partitioned. CMS can never write to real OS data sets. CMS reads and writes VSAM data sets.
data definition (DD) card	The FILEDEF command allows you to perform the functions of the DD statement to specify device types and output file dispositions.
data set control block (DSCB)	Information about a CMS disk file is contained in a file status table (FST).
EXEC card	To execute a program in CMS you specify only the name of the program if it is an EXEC, MODULE file, or CMS command. To execute TEXT files, use the LOAD and START commands.
job control language (JCL)	CMS and user-written commands perform the functions of JCL.
link-editing	The CMS LKED command creates LOADLIB libraries from CMS TEXT files and/or OS object modules. The CMS LOAD command loads TEXT files into virtual storage, and resolves external references; the GENMOD command creates MODULE files.
load module	Load modules are members of CMS LOADLIB libraries. LOADLIB members are loaded, relocated, and executed by the OSRUN command, and LOADLIB members are loaded and relocated by the NUCXLOAD command. Also, LOADLIB members are referenced by the LINK, LOAD, ATTACH and XCTL macros.
object module	Language compiler output is placed in CMS files with a filetype of TEXT.
partitioned data set	CMS MACLIBs, TXTLIBs, and LOADLIBs are the only CMS files that resemble partitioned data sets.
STEPCAT, JOBCAT	VSAM catalogs can be assigned for jobs or job steps in CMS by using the special ddnames IJSYSCT and IJSYSUC when identifying catalogs.
STEPLIB, JOBLIB	The GLOBAL command establishes macro, text, and LOADLIB libraries; you can indirectly provide job libraries by accessing and releasing CMS disks that contain the files and programs you need.
utility program	Functions similar to those performed by the OS utility programs are provided by CMS commands.
volume table of contents (VTOC)	The list of files on a CMS disk is contained in a file directory.

Figure 18. OS Terms and CMS Equivalents

Using OS Data Sets in CMS

You can have OS disks defined in your virtual machine configuration; they may be either entire disks or minidisks: their size and extent depends on their VM/SP directory entries. You can use partitioned and sequential data sets on OS disks in CMS. If you want, you can create CMS files from your OS data sets. If you have data sets on OS disks, you can read them from programs you execute in CMS, but you cannot update them. The CMS commands that recognize OS data sets on OS disks are listed in Figure 19.

Command	Operation
ACCESS	Makes the OS disk containing the data set available to your CMS virtual machine.
ASSEMBLE	Assembles an OS source program under CMS.
DDR	Copies an entire OS disk to tape.
DLBL	Defines OS data sets for use with access method services and VSAM files for program input/output.
FILEDEF	Defines the OS data set for use under CMS by associating an OS ddname with an OS data set name. Once defined, the data set can be used by an OS program running under CMS and can be manipulated by the other commands that support OS functions.
GLOBAL	Makes macro libraries or LOADLIB libraries available to CMS. You can prepare an OS library for reference by the GLOBAL command by issuing a FILEDEF command for the data set and giving the data set the appropriate filetype of MACLIB or LOADLIB.
LKED	Creates CMS LOADLIB libraries from CMS TEXT files and/or OS object modules.
LISTDS	Lists information describing OS data sets residing on OS disks.
MOVEFILE	Moves data records from one device to another device. Each device is specified by a ddname, which must have been defined via FILEDEF. You can use the MOVEFILE command to create CMS files from OS data sets.
NUCXLOAD	Loads, relocates, and establishes as a nucleus extension a load module either from a CMS LOADLIB, an OS module library on an OS formatted disk.
OSRUN	Loads, relocates, and executes a load module either from a CMS LOADLIB or from an OS module library on an OS formatted disk.
QUERY	Lists (1) the files that have been defined with the FILEDEF and DLBL commands (QUERY FILEDEF, QUERY DLBL), or (2) the status of OS disks attached to your virtual machine (QUERY DISK, QUERY SEARCH).
RELEASE	Releases an OS disk you have accessed (via ACCESS) from your CMS virtual machine.
STATE	Verifies the existence of an OS data set on a disk. Before STATE can verify the existence of the data set, you must have defined it (via FILEDEF).

Figure 19. CMS Commands that Recognize OS Data Sets on OS Disks

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.

Figure 22 on page 189 shows the OS macro functions that are partially or completely simulated, as defined by SVC number.

OS Simulated Data Sets

If you want to test programs in CMS that create or modify OS data sets, you can write "OS simulated data sets." These are CMS files that are maintained on CMS disks, but in OS format rather than in CMS format. Since they are CMS files, you can edit, rename, copy, or manipulate them just as you would any other CMS file. Since they are in OS-simulated format, files with variable-blocked records may contain block and record descriptor words so that the access methods can manipulate them properly.

The files that you create from OS programs do not necessarily have to be OS simulated data sets. You can create CMS files. The format of an output file depends on how you specify the filemode number when you issue the FILEDEF command to identify the file to CMS. If you specify the filemode number as 4, CMS creates a file that is in OS simulated data set format on a CMS disk. If you want to read an OS simulated data set that is variable blocked or fixed blocked, rename the data set with a filemode number of 4. CMS OS simulation routines are then able to read short blocks that are not filled with records.

CMS can read and write OS simulated data sets using the BDAM, BPAM, BSAM, and QSAM access methods. See "Access Method Support" on page 199 for a description of these access methods.

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 the *VM/SP System Messages and Codes* under DMS message 120S.

| *Note:* Results may be unpredictable if two DCBs access the same data set
| at the same time.

Restrictions for Reading OS Data Sets

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

- Read-password-protected data sets are not read.
- RACF password protection is ignored.
- BDAM and ISAM data sets are not read.
- Multivolume data sets are read as single-volume data sets. End-of-volume is treated as end-of-file and there is no end-of-volume switching.
- Keys in data sets with keys are ignored; only the data is read.
- User labels in user-labeled data sets are bypassed. See "Tape Labels in CMS" in the *VM/SP CMS User's Guide* for details.
- Results may be unpredictable if two DCBs access the same data set at the same time.
- An Indexed VTOC on an OS disk is read the same as a standard OS VTOC since there is no special support in CMS for this.

The following restrictions apply when you are reading OS data sets from tapes under CMS:

- Read-password-protected data sets are read.
- RACF password protection is ignored.
- User labels in user-labelled data sets are bypassed. See "Tape Labels in CMS" in the *VM/SP CMS User's Guide* for details.
- Results may be unpredictable if two DCBs access the same data set at the same time.

The ACCESS Command

Before CMS can read an OS data set 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 *VM/SP CMS Command Reference.* You must not specify options or file identification when accessing an OS disk.

The FILEDEF Command

Whenever you execute an OS program under CMS that has input and/or output files or you need to read an OS data set onto a CMS disk, you must first identify the files to CMS with the FILEDEF command. The FILEDEF command in CMS performs the same functions as the data definition (DD) card in OS job control language (JCL). The data definition card describes the input and output files.

When you enter the FILEDEF command, you specify:

- The ddname
- The device type
- A file identification, if the device type is DISK
- Type of label on your tape file, if tape label processing is specified
- Options (if necessary).

Some guidelines for entering these specifications follow.

Specifying the ddname

If the FILEDEF command is issued for a program input or output file, the ddname must be the same as the ddname or file name specified for the file in the source program. For example, you can have an assembler language source program that contains the line:

```
INFILE  DCB  DDNAME=INPUTDD,MACRF=GL,DSORG=PS,RECFM=F,
             LRECL=80
```

For a particular execution of this program, you want to use as your input file a CMS file on your A-disk that is named MYINPUT FILE. You must then issue a FILEDEF for this file before executing the program:

```
filedef inputdd disk myinput file a1
```

If the input file you want to use is on an OS disk accessed as your C-disk and it has a data set name of PAYROLL.RECORDS.AUGUST, then your FILEDEF command might be:

```
filedef inputdd c1 dsn payroll.records.august
```

Specifying the Device Type

For input files, the device type you enter on the FILEDEF command indicates the device from which you want records read. It can be DISK, TERMINAL, READER (for input from real cards or virtual cards), or TAPn (for tape). Using the above example, if your input file is to be read from your virtual card reader, the FILEDEF command might be as follows:

```
filedef inputdd reader
```

Or, if you were reading from a tape attached to your virtual machine at virtual address 181 (TAP1):

```
filedef inputdd tap1
```

For output files, the device you specify can be DISK, PRINTER, PUNCH, TAPn (tape), or TERMINAL.

If you do not want any real I/O performed during the execution of a program for a disk input or output file, you can specify the device type as DUMMY:

```
filedef inputdd dummy
```

Entering File Identifications

If you are using a CMS disk file for your input or output, specify:

```
filedef ddname disk filename filetype filemode
```

Note: If * is used for the filemode of an output file, unpredictable results may occur.

The filemode field is optional; if you do not specify it, your A-disk is assumed.

If you want an output file to be constructed in OS simulated data set format, you must specify the filemode number as 4. For example, if a program contains a DCB for an output file with a ddname of OUTPUTDD and you are using it to create a CMS file named DAILY OUTPUT on your B-disk, specify:

```
filedef outputdd disk daily output b4
```

If your input file is an OS data set on an OS disk, you can identify it in several ways:

- If the data set name has only two qualifiers, for example HEALTH.RECORDS, you can specify:

  ```
  filedef inputdd disk health records b1
  ```

- If it has more than two qualifiers, you can use the DSN keyword and enter:

```
filedef inputdd b1 dsn health records august 1974
                   -- or --
filedef inputdd b1 dsn health.records.august.1974
```

Or you can request a prompt for a complete data set name:

```
filedef inputdd b1 dsn ?
Enter data set name:
health.records.august.1974
```

Note: When you enter a data set name using the DSN keyword either with or without a request for prompting, you should omit the device type specification of DISK, unless you want to assign a CMS file identifier, as in the example below.

- You can also relate an OS data set name to a CMS file identifier:

  ```
  filedef inputdd disk ossim file c1 dsn monthly records
                       -- or --
  filedef inputdd disk ossim file c1 dsn monthly.records
  ```

  Then you can refer to the OS data set MONTHLY.RECORDS by using the CMS file identifier, OSSIM FILE:

  ```
  state ossim file c
  ```

When you do not issue a FILEDEF command for a program input or output file or if you enter only the ddname and device type on the FILEDEF command, such as:

```
filedef oscar disk
```

then CMS issues a default file definition, as follows:

```
FILEDEF ddname DISK FILE ddname A1
```

where ddname is the ddname you assigned in the DDNAME operand of the DCB macro in your program or on the FILEDEF command. For example, if you assign a ddname of OSCAR to an output file and do not issue a FILEDEF command before you execute the program, then the CMS file FILE OSCAR A1 is created when you execute the program. If the filetype of a CMS input file, FILE ddname A1, is the same as the assigned DDNAME, the file can be identified by a default file definition. Even though an input file can be defined explicitly or by default, if an attempt is made to read the file and the file is not found, unpredictable results may occur.

Specifying CMS Tape Label Processing

You can use the label operands on the FILEDEF command to indicate that CMS tape label processing is not desired (this is the default). If CMS tape label processing is desired you can use the label operands on the FILEDEF command to indicate the types of labels on your tape. See the *VM/SP CMS User's Guide* for a description of CMS tape label processing.

Specifying Options

The FILEDEF command has many options; those mentioned below are a sampling only. For complete descriptions of all the options of the FILEDEF command, see the *VM/SP CMS Command Reference.*

Supplying File Format Information: If you are using the FILEDEF command to relate a data control block (DCB) in a program to an input or output file, you may need to supply some of the file format information on the FILEDEF command line, such as the block size (BLOCK), record length (LRECL), record format (RECFM), and data set organization (DSORG). For example, if you have coded a DCB macro for an output file as follows:

```
OUTFILE  DCB   DDNAME=OUT,MACRF=PM,DSORG=PS
```

then, when you are issuing a FILEDEF for this ddname, you must specify the format of the file. To create an output file on disk blocked in OS simulated data set format, you could issue:

```
filedef out disk myoutput file a4 (recfm fb lrecl 80 block 1600
```

To punch the output file onto cards, you would issue:

```
filedef out punch (lrecl 80 recfm f
```

You can omit file format information on the FILEDEF command line whenever it is supplied on the DCB macro or whenever your file exists on an OS disk. For existing CMS disk files, format information is required only if you want OS-simulated data set formats other than F or V. 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 order in which the DCB fields are filled follows:

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. The 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.

Keeping File Definitions: Usually, when you execute one of the language processors, all existing file definitions are cleared. If the development of a program requires you to recompile and re-execute it frequently, you might want to use the PERM option when you issue file definitions for your input and output files. For example:

```
cp spool punch to *
filedef indd disk test file a1 (lrecl 80 perm
filedef outdd punch (lrecl 80 perm
```

In this example, since you spooled your virtual punch to your own virtual card reader, output files are placed in your virtual reader. You can either read or delete them.

All file definitions issued with the PERM option stay in effect until you log off, specifically clear those definitions, or redefine them:

```
filedef indd clear
filedef outdd tap1 (lrecl 80
```

In the above example, the definition for INDD is cleared; OUTDD is redefined as a tape file.

When you issue the command:

```
filedef * clear
```

all file definitions are cleared, except those you enter with the PERM option.

When a program abends, or when you issue the HX Immediate command, all file definitions are cleared, including those entered with the PERM option.

Adding Records to a File: When you issue a FILEDEF command for an output file and assign it a CMS file identifier that is identical to that of an existing CMS file, the existing file is replaced by the new output file if anything is written to that ddname. If you want, instead, to have new records added to the bottom of the existing file, you can use the DISP MOD option:

```
filedef outdd disk new update a1 (disp mod
```

The file must be on a disk accessed as read/write. Note that an extension of a disk is read/only. When adding new records using the DISP MOD option, use an editor to delete the end-of-file (EOF) mark at the end of the existing file for fixed-block (FB) OS simulated files (filemode of A4).

Specifying a Member Name of a Data Set: If the file you want to read is a member of an OS partitioned data set (or a CMS MACLIB or TXTLIB), you can use the MEMBER option to specify the member name. For example:

```
filedef test c dsn sys1.maclib (member test
```

defines the member TEST from the OS macro library SYS1.MACLIB.

Receiving Control during I/O Operation: The AUXPROC option is valid only when FILEDEF is executed by an internal program call. It cannot be entered as a terminal command. 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 purpose registers. This routine must also perform the save operation. DMSSEB does not provide the address of a save area in general purpose register 13, as is usually the case. When control returns to DMSSEB, the general purpose registers must be restored to their original values. Control is returned to DMSSEB by branching to the address contained in general purpose 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:

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=65,536	I/O performed by AUXPROC routine with zero residual count.

Passing Information to the DMSTVI Routine: An interface routine, DMSTVI, can be used to give control to a different multivolume switching routine than the one supplied with VM (DMSTVS) or a tape management system.

Use the new SYSPARM option to pass information not included on the FILEDEF or LABELDEF command to the DMSTVI routine.

When DMSTVI is called, the general-purpose registers contain the following information:

GPR 1	= Address of a parameter list defined by the TVISECT DSECT
GPR 14	= Return address
GPR 15	= Entry point address

The calling routine saves and restores the register contents.

When DMSTVI gets control, it must check the call function keyword in the register 1 PLIST. The call function keyword identifies the function being processed when DMSTVI is called. DMSTVI should use the information in the PLIST to build a command or to invoke the tape volume switching routine or tape management system.

When DMSTVI is called during FILEDEF processing, only the call function (SYSPARM) and the SYSPARM string address and length field are filled in. The other fields are set to zeroes.

Because DMSTVI gets control during OPEN macro processing before any I/O is done, you do not have to mount a tape before OPEN is issued. The interface routine can mount the tape before returning control to OPEN macro processing.

If you specify only the first volid of a multivolume tape and the end of the first volume is reached, DMSTVI gets control with a call function of 'EOV' and a volid of SCRATC. The tape management system can mount the next volume if it knows what tape is currently mounted on the drive and the volid of the next volume in the series.

A 44 character fileid can now be entered with the LABELDEF command. The 44 character fileid is passed to DMSTVI during OPEN, EOV, and CLOSE macro processing. DMSTVI should check the TVISCRAT field in the register 1 PLIST to determine if a tape was requested from the tape management system. DMSTVI checks the TVISCRAT field by giving a fileid.

If the TVISCRAT field contains 'SCRATCH', a scratch tape was requested. If this field contains 'NOSCRATC', a scratch was not requested -- 'SCRATC' was put in TVIVOLID as a default. If a fileid is also specified (TVIFILID), a tape containing this fileid was requested. If you want to mount a tape by specifying just the fileid, you should not specify any volid on FILEDEF or LABELDEF (including 'SCRATCH').

If no fileid is specified on the LABELDEF command, the TVIFID field in the register 1 PLIST contains all zeros. The system uses the ddname (TVIFILE) as the default.

DMSTVI must return to the calling routine when processing is complete.

Creating CMS Files from OS Data Sets

If you have data sets on OS disks, on tapes, or on cards, you can copy them into CMS files so that you can edit, modify, or manipulate them with CMS commands. The CMS MOVEFILE command copies OS (or CMS) files from one device to another. You can move data sets from any valid input device to any valid output device.

Before using the MOVEFILE command, you must define the input and output data sets or files and assign them ddnames using the FILEDEF command. If you use the ddnames INMOVE and OUTMOVE, then you do not need to specify the ddnames when you issue the MOVEFILE command. For example, the following sequence of commands copies a CMS disk file into your virtual card punch:

```
filedef inmove disk diskin file a1
filedef outmove punch
movefile
```

The result of these commands is effectively the same as if you had issued the command:

```
punch diskin file (noheader
```

The example does, however, illustrate the basic relationship between the FILEDEF and MOVEFILE commands. In addition to the MOVEFILE command, if the OS input data set is on tape or cards, you can use the TAPPDS or READCARD command to create CMS files. These are also discussed below.

Note: The MOVEFILE command does not support data containing spanned records. In addition, when copying a variable length data set (RECFM=V or VB) from an OS disk to a CMS disk, the logical record length (LRECL) of the file that is created on the CMS disk is equal to the size of the largest record in the data set being copied. If the file that is being created has a filemode of 4, the logical record length will be equal to the LRECL of the largest record plus 8 bytes. The actual LRECL of the new file can be determined by using the CMS LISTFILE command.

Copying Sequential Data Sets from Disk

The MOVEFILE command copies a sequential OS disk data set from a read-only OS disk into an integral CMS file on a CMS read/write disk. You use FILEDEF commands to identify the input file disk mode and data set name:

```
filedef inmove c1 dsn sales.manual
```

the CMS output file's disk location and fileid:

```
filedef outmove disk sales manual a1
```

and then you issue the MOVEFILE command:

```
movefile
```

Copying Partitioned Data Sets From Disk

The MOVEFILE command can copy partitioned data sets (PDS) into CMS disk files and create separate CMS files for each member of the data set. You can have the entire data set copied, or you can copy only a selected member. For example, if you have a partitioned data set named ASSEMBLE.SOURCE whose members are individual assembler language source files, your input file definition might be:

```
filedef inmove c1 dsn assemble source
           -- or --
filedef inmove c1 dsn assemble.source
```

To create individual CMS ASSEMBLE files, you would issue the output file definition as:

```
filedef outmove disk qprint assemble a1
```

Then use the PDS option of the MOVEFILE command:

```
movefile (pds
```

When the CMS files are created, the filetype on the output file definition is used for the filetype and the member names are used instead of the CMS filename you specified.

If you want to copy only a single member, you can use the MEMBER option of the FILEDEF command:

```
filedef inmove disk assemble source c (member qprint
```

and omit the PDS option on the MOVEFILE command:

```
movefile
```

The following figure summarizes the various ways that you can create CMS files from OS data sets.

Input File: An OS Sequential Data Set Named: COMPUTE.TEST.RECORDS

Source	CMS Command Examples	CMS Output File
Disk:	filedef indd c1 dsn compute test records	COMPUTE RECORDS A1
OS R/O	filedef outdd disk compute records a1	
C-disk	movefile indd outdd	
Tape:	filedef inmove tap1 (lrecl 80	TEST RECORDS A1
181	filedef outmove disk test records a1	
	movefile	
	tappds newtest compute (nopds	NEWTEST COMPUTE A1
Cards:	filedef cardin reader	
	filedef diskout disk compute cards a1	COMPUTE CARDS A1
	movefile cardin diskout	
	readcard compute test	COMPUTE TEST A1

Input file: OS Partitioned Data Set Named: TEST.CASES
Members named: SIMPLE, COMPLEX, MIXED

Source	CMS Command Examples	CMS Output File(s)
Disk:	filedef infile c1 dsn test cases	SIMPLE TESTCASE A1
OS R/O	filedef outfile disk new testcase a1	COMPLEX TESTCASE A1
C-disk	movefile infile outfile (pds	MIXED TESTCASE
	filedef in c1 dsn test cases (member simple	FILE RUN A1
	filedef run disk	
	movefile in run	
Tape:	tappds * testrun (tap2	SIMPLE TESTRUN A1
182		COMPLEX TESTRUN A1
		MIXED TESTRUN A1

Figure 20. Creating CMS Files from OS Data Sets

Using CMS Libraries

CMS provides three types of libraries to aid in OS program development:

- Macro libraries contain macro definitions and/or copy files.
- Text, or program libraries contain relocatable object programs (compiler output).
- LOADLIB libraries contain link edit files (load modules).

These CMS libraries are like OS partitioned data sets; each has a directory and members. Since they are not like other CMS files, you create, update, and use them differently than you do other CMS files. Although these library files are similar in function to OS partitioned data sets, OS macros should not be used to update them. Macro libraries are discussed below; text libraries are discussed under "TEXT Libraries (TXTLIBs)" on page 181, and LOADLIB libraries are discussed under "OS Module Libraries and CMS LOADLIBS" on page 183.

Macro Libraries (MACLIBs)

A CMS macro library has a filetype of MACLIB. You can create a MACLIB from files with filetypes of MACRO or COPY. A MACRO file may contain macro definitions. COPY files contain predefined source statements.

The MACLIB Command

The MACLIB command performs a variety of functions. You use it to:

- Create the MACLIB (GEN function).
- Add, replace, or delete members (ADD, REP, and DEL functions).
- Compress the MACLIB (COMP function).
- List the contents of the MACLIB (MAP function).

Descriptions of these MACLIB command functions follow.

Creating a Macro Library: The GEN (generate) function creates a CMS macro library from input files specified on the command line. The input files must have filetypes of either MACRO or COPY. For example:

```
maclib gen osmac access time put regequ
```

creates a macro library with the file identifier OSMAC MACLIB A1 from macros existing in the files with the file identifiers:

$$\text{ACCESS}\left\{\begin{matrix}\text{MACRO}\\\text{COPY}\end{matrix}\right\},\ \text{TIME}\left\{\begin{matrix}\text{MACRO}\\\text{COPY}\end{matrix}\right\},\ \text{PUT}\left\{\begin{matrix}\text{MACRO}\\\text{COPY}\end{matrix}\right\},\ \text{and REGEQU}\left\{\begin{matrix}\text{MACRO}\\\text{COPY}\end{matrix}\right\}$$

If a file named OSMAC MACLIB A1 already exists, it is erased.

Assume that the files ACCESS MACRO, TIME COPY, PUT MACRO, and REGEQU COPY exist and contain macros in the following form:

```
ACCESS MACRO        COPY            PUT MACRO       REGEQU COPY

    GET         *COPY TTIMER           PUT              XREG
                 TTIMER
    PUT         *COPY STIMER                            YREG
                 STIMER
```

The resulting file, OSMAC MACLIB A1, contains the members:

```
GET       STIMER
PUT       PUT
TTIMER    REGEQU
```

The PUT macro, which appears twice in the input to the command, also appears twice in the output. The MACLIB command does not check for duplicate macro names. If, at a later time, the PUT macro is requested from OSMAC MACLIB, the first PUT macro encountered in the directory is used.

When COPY files are added to MACLIBs, the name of the library member is taken from the name of the COPY file or from the *COPY statement, as in the file TIME COPY, above.

Note: Although the file REGEQU COPY contained two macros, they were both included in the MACLIB with the name REGEQU. When the input file is a MACRO file, the member name(s) are taken from macro prototype statements in the MACRO file.

Adding a Member to a Macro Library: The ADD function appends new members to an existing macro library. For example, assume that OSMAC MACLIB A1 exists as created in the example in the explanation of the GEN function and the file DCB COPY exists as follows:

```
*COPY DCB
 DCB macro definition
*COPY DCBD
 DCBD macro definition
```

If you issue the command:

```
maclib add osmac dcb
```

the resulting OSMAC MACLIB A1 contains the members:

```
GET       PUT
PUT       REGEQU
TTIMER    DCB
STIMER    DCBD
```

Replacing a Member of a Macro Library: The REP (replace) function deletes the directory entry for the macro definition in the files specified. It then appends new macro definitions to the macro library and creates new directory entries. For example, assume that a macro library MYMAC MACLIB contains the members ALPHA, BETA, and SIGMA, and that the following command is entered:

```
maclib rep mymac alpha sigma
```

The files represented by file identifiers ALPHA MACRO and SIGMA MACRO each have one macro definition. After execution of the command, MYMAC MACLIB contains members with the same names as before, but the contents of ALPHA and SIGMA are different.

Deleting a Member of a Macro Library: The DEL (delete) function removes members from the macro library directory and compresses the directory so there are no unused entries. The macro definition still occupies space in the library, but since no directory entry exists, it cannot be accessed or retrieved. If you attempt to delete a macro for which two macro definitions exist in the macro library, only the first one encountered is deleted. For example:

```
maclib del osmac get put ttimer dcb
```

deletes macro names GET, PUT, TTIMER, and DCB from the directory of the macro library named OSMAC MACLIB. Assume that OSMAC exists as in the ADD function example. After the above command, OSMAC MACLIB contains the following members:

```
STIMER
PUT
REGEQU
DCBD
```

Compressing a Macro Library: Execution of a MACLIB command with the DEL or REP functions can leave unused space within a macro library. The COMP (compress) function removes any macros that do not have directory entries. This function uses a temporary file named MACLIB CMSUT1. For example, the command:

```
maclib comp mymac
```

compresses the library MYMAC MACLIB.

Listing Information about Members of a Macro Library: The MAP function creates a list containing the name of each macro in the directory, the size of the macro, and its position within the macro library. If you want to display a list of the members of a MACLIB at the terminal, enter the command:

```
maclib map mylib (term
```

The default option, DISK, creates a file on your A-disk, which has a filetype of MAP and a filename corresponding to the filename of the MACLIB. If you specify the PRINT option, the list is spooled to your virtual printer as well as being written onto disk.

Note: The DISK, PRINT, and TERM options erase the old MAP file.

You can also retrieve information for specific members of the library by indicating the member names following the MAP operand. For example:

```
maclib map mylib swerve yield
```

returns the MAP output for only members SWERVE and YIELD of MYLIB MACLIB.

If you want to place that information in the program stack, use the STACK option of the MAP operand. The information can be stacked FIFO (first-in first-out) or LIFO (last-in first-out). The default order when STACK is specified alone is FIFO. The options STACK, STACK FIFO, and FIFO are equivalent. The options STACK LIFO and LIFO are equivalent. For example:

```
maclib map mylib neutral reverse (stack fifo
```

stacks in the program stack, the MAP output for the NEUTRAL and REVERSE members of MYLIB in first-in first-out order.

See "The MACLIST Command" on page 176 for more information on listing members of a MACLIB.

Manipulating MACLIB Members

The following CMS commands recognize MACLIBs and have a MEMBER option:

- XEDIT (to create and/or edit a specific member).
- PUNCH (to punch a member)
- FILEDEF (to establish a file definition for a member)
- PRINT (to print a member)
- TYPE (to display a member at the terminal)

You can use the editor to create MACRO and COPY files and then use the MACLIB command to place the files in a library. Once they are in a library, you can erase the original files, or you can edit a member of a CMS library using the XEDIT command with the MEMBER option. For example, entering the command:

```
xedit mylib maclib a1 (member swerve
```

If the SWERVE member does not exist in that library, a new file is created with a fileid of SWERVE MEMBER A1. If SWERVE is an existing member of MYLIB MACLIB, you can edit the file.

You can also select members of a specific CMS library to edit from your MACLIST (invoked by the MACLIST command).

Note: You cannot create a new MACLIB using the MEMBER option of the XEDIT command. You must use the MACLIB command with the GEN option to create a new MACLIB.

To extract a member from a macro library, you can use either the PUNCH or the MOVEFILE command. If you use the PUNCH command, you can spool your virtual card punch to your own virtual reader:

```
cp spool punch to *
```

Then punch the member:

```
punch testmac maclib (member get noheader
```

and read it back onto disk:

```
readcard get macro
```

In the above example, the member was punched with the NOHEADER option of the PUNCH command, so that a name could be assigned on the READCARD command line. If a header card had been created for the file, it would have indicated the filename and filetype as GET MEMBER.

If you use the MOVEFILE command, you must issue a file definition for the input member name and the output macro or copy name before entering the MOVEFILE command:

```
filedef inmove disk testcopy maclib (member enter
filedef outmove disk enter copy a
movefile
```

This example copies the member ENTER from the macro library TESTCOPY MACLIB into a CMS file named ENTER COPY.

When you use the PUNCH or MOVEFILE commands to extract members from CMS MACLIBs, each member is followed by a // record, which is a MACLIB delimiter. You can edit the file and use the DELETE subcommand to delete the // record.

If you want to move the complete MACLIB to another file, use the COPYFILE command.

To print a single member or all members of a macro library, use the CMS PRINT command with the MEMBER option. To display on the terminal a single member or all members of a macro library, use the CMS TYPE command with the MEMBER option.

The MACLIST Command

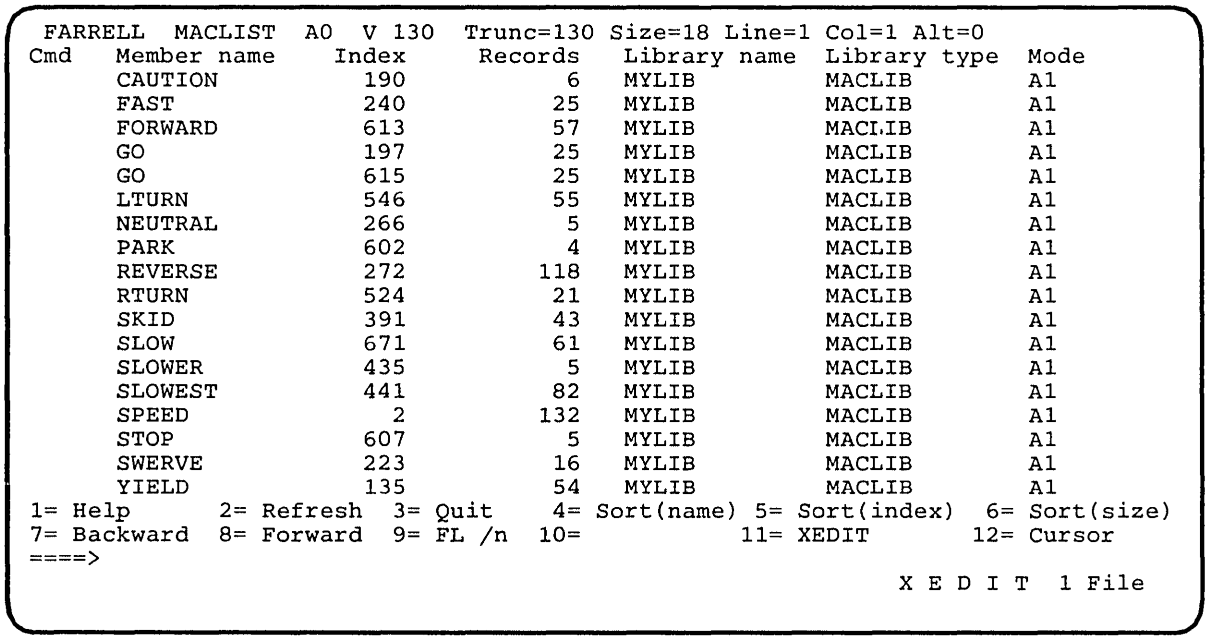
The MACLIST command displays a list of all members in a specified macro library. MACLIST provides you with an easy way to select and edit CMS maclib members. CMS commands can be issued against the members directly from the displayed list. The commands execute when you press the ENTER key.

In the MACLIST environment, information that is normally provided by the MACLIB command (with the MAP option) is displayed under the control of the System Product editor. You can use XEDIT subcommands to manipulate the list itself.

The following MACLIST screen was created by issuing the MACLIST command as follows:

```
maclist mylib
```

Note that the members are sorted alphabetically by member name. Members with the same name are then sorted by index number (least to greatest).



```
 FARRELL  MACLIST  A0  V 130  Trunc=130 Size=18 Line=1 Col=1 Alt=0
Cmd   Member name   Index     Records   Library name  Library type  Mode
      CAUTION         190           6   MYLIB         MACLIB        A1
      FAST            240          25   MYLIB         MACLIB        A1
      FORWARD         613          57   MYLIB         MACLIB        A1
      GO              197          25   MYLIB         MACLIB        A1
      GO              615          25   MYLIB         MACLIB        A1
      LTURN           546          55   MYLIB         MACLIB        A1
      NEUTRAL         266           5   MYLIB         MACLIB        A1
      PARK            602           4   MYLIB         MACLIB        A1
      REVERSE         272         118   MYLIB         MACLIB        A1
      RTURN           524          21   MYLIB         MACLIB        A1
      SKID            391          43   MYLIB         MACLIB        A1
      SLOW            671          61   MYLIB         MACLIB        A1
      SLOWER          435           5   MYLIB         MACLIB        A1
      SLOWEST         441          82   MYLIB         MACLIB        A1
      SPEED             2         132   MYLIB         MACLIB        A1
      STOP            607           5   MYLIB         MACLIB        A1
      SWERVE          223          16   MYLIB         MACLIB        A1
      YIELD           135          54   MYLIB         MACLIB        A1
1= Help      2= Refresh  3= Quit    4= Sort(name) 5= Sort(index)  6= Sort(size)
7= Backward  8= Forward  9= FL /n  10=           11= XEDIT       12= Cursor
====>
                                                           X E D I T  1 File
```

Figure 21. Sample MACLIST Screen

Finding Members in Your MACLIST List: If there are many members in the maclib, the list may take up more than one screen. To find a member in your MACLIST list, you can do any of the following:

- Scroll through the list using the PF keys.

  PF7 Scrolls backward one full screen.

  PF8 Scrolls forward one full screen.

- Rearrange the list using one of the following PF keys:

  PF4 Sorts the list by member name. This is how the list is initially arranged.

  PF5 Sorts the list by index (largest first). The most recently updated members have a greater number.

  PF6 Sorts the list by size (largest to smallest).

- Use the XEDIT subcommand LOCATE if you know the member name that you are looking for.

- Rearrange the list by entering one of the following synonyms on the command line:

  SINDEX Sorts the list by index (greatest to least) within a library.

SLIB	Sorts the list alphabetically by library fileid.
SNAME	Sorts the list alphabetically by member name. This is how the list is initially arranged.
SSIZE	Sorts the list by member size (number of records, greatest to least).

Entering Commands in the MACLIST Environment: You can type commands that operate on member names in the list directly on the lines of the MACLIST display. When you press the ENTER key, all commands typed on the lines in the file displayed on the current screen are executed. Symbols can be used to represent operands in the command to be executed. Symbols are needed if the command to be executed has operands or options that follow the fileid. For example to issue the PRINT command for this member of your MACLIST:

```
      NEUTRAL      266         5   MYLIB       MACLIB    A1
```

type directly on the line that contains this member as follows:

```
print /EUTRAL      266         5   MYLIB       MACLIB    A1
```

and then press the ENTER key. Refer to the MACLIST command in the *VM/SP CMS Command Reference* for more information about using symbols in MACLIST.

Another way to issue commands that make use of member names displayed is to move the current line to the first (or only) member you want the command to use. Then issue an EXECUTE command (in the form "EXECUTE lines command") from the XEDIT command line. This method may be used on both display and typewriter terminals. You can also enter commands from the MACLIST command line.

Editing a Maclib Member: The MACLIST command allows you to select and edit a CMS maclib member from the list. To edit a member, position the cursor on the line that contains the member to be edited and press the PF11 key. Otherwise, you can edit a CMS maclib member by using the XEDIT command with the MEMBER option. For example, to edit the SWERVE member of MYLIB maclib, enter:

```
xedit mylib maclib a1 (member swerve
```

If the SWERVE member did not exist in MYLIB MACLIB, a new file is created with a fileid of SWERVE MEMBER A1.

Adding and Replacing Maclib Members: When the MEMBER option is specified for the XEDIT command for a member that does not exist in the library, a new file is created with the fileid of "membername MEMBER fm."

If the MEMBER option is specified on the XEDIT command for an existing member of a library, the member is read into a file called "membername MEMBER fm" for you to edit.

When you issue FILE or SAVE for the new or changed member, the library directory is updated. The new or changed member and the updated library directory are added to the end of the library. If the directory already contains a member with the same name as the one being saved, the old entry is blanked out, so that the updated member replaces the old version.

Deleting Maclib Members: Use the DISCARD command to delete a member from a library. DISCARD is equivalent to the CMS command MACLIB DEL. DISCARD can either be typed in the command area of the line that describes the member you want discarded, or it can be entered from the command line (at the bottom of the screen). DISCARD can only be used while in the FILELIST, RDRLIST, MACLIST, and PEEK command environments.

Setting MACLIST Defaults: When XEDIT is invoked by the MACLIST command to display the list, the default XEDIT macro, PROFMLST XEDIT, is executed. If you want to invoke a different XEDIT macro, you can specify the PROFILE option with the MACLIST command. For example, to invoke MACLIST with the MYMCLST XEDIT macro, enter

```
maclist mylib (profile mymclst
```

You can do the same with the COMPACT and NOCOMPACT options of the MACLIST command.

If you are using an alternate profile most of the time, you may change the default profile with the DEFAULTS command. For example:

```
defaults set maclist profile mymclst
```

Entering the DEFAULTS command with no options provides you with the status of defaults currently in effect. For example, entering

```
defaults
```

after changing the XEDIT macro, returns the following information:

```
| The following default options have been set:

| Filelist options = PROFILE PROFFLST NOFILELIST
| Help options = SCREEN BRIEF ALL
| Maclist options = PROFILE MYMCLST NOCOMPACT
| Note options = PROFILE PROFNOTE SHORT LOG NOACK NOTEBOOK ALL
| Peek options = PROFILE PROFPEEK FROM 1 FOR 200
| Rdrlist options = PROFILE PROFRLST
| Receive options = LOG OLDDATE NOTEBOOK ALL
| Sendfile options = NEW TYPE NOFILELIST LOG NOACK
| Tell options = MSGCMD MSG

| To change any default options enter DEFAULTS Set Cmdname Opt1 <Opt2..>
```

The GLOBAL Command

When you want to assemble or compile a source program that uses macro or copy definitions, you must ensure that the library containing the code is identified before you invoke the assembler. Otherwise, the library is not searched. You identify libraries to be searched using the GLOBAL command. For example, if you have two MACLIBs that contain your private macros and copy files whose names are TESTMAC MACLIB and TESTCOPY MACLIB, you would issue the command:

```
global maclib testmac testcopy
```

The libraries you specify on a GLOBAL command line are searched in the order you specify them. A GLOBAL command remains in effect for the remainder of your terminal session, until you issue another GLOBAL MACLIB command or IPL CMS again. To find out what macro libraries are currently available for searching, issue the command:

```
query maclib
```

You can reset the libraries or the search order by reissuing the GLOBAL command.

System MACLIBs

The macro libraries that are on the system disk contain CMS and OS assembler language macros you may want to use in your programs. The MACLIBs are:

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

  Note: When assembling programs that use CMS macros, both of these libraries should be identified via the GLOBAL command. DMSSP should precede CMSLIB in the search order.
- OSMACRO MACLIB contains the OS macros that CMS supports or simulates or those that require no CMS support.

- OSMACRO1 MACLIB contains the macros CMS does not support or simulate. (You can assemble programs in CMS that contain these macros, but you must execute them in an OS virtual machine.)

- OSVSAM MACLIB contains the subset of supported OS/VSAM macros.

- TSOMAC MACLIB contains TSO macros.

- DOSMACRO MACLIB contains macros used internally in CMS/DOS.

  Note: The DOSMACRO MACLIB contains macros used internally by CMS/DOS system routines. These macros should not be used in user written programs.

To obtain a list of macros in any of these libraries, use either the MACLIST command or the MACLIB command with the MAP function. In the MACLIST environment, you can issue CMS commands against the members directly from the displayed list. You can find more information about the MACLIST command in the *VM/SP CMS Command Reference.*

TEXT Libraries (TXTLIBs)

You may want to keep your TEXT files in text libraries. These files have a filetype of TXTLIB. You can create a TXTLIB from files with a filetype of TEXT. Like MACLIBs, TXTLIBs have a directory and members.

The TXTLIB Command

TXTLIBs are created and modified by the TXTLIB command, which has functions similar to the MACLIB command:

- Create the TXTLIB (GEN function).
- Add members to the TXTLIB (ADD function).
- Delete members of the TXTLIB and compress the TXTLIB (DEL function).
- List the members of the TXTLIB (MAP function).

There is no REP function. You must use a DEL followed by an ADD to replace an existing member.

Creating a TXTLIB: The TXTLIB command with the FILENAME option specified reads the object files as it writes them into the library and creates a directory entry for each filename. If you have a TEXT file named MYPROG, which has an entry point named BEGIN, create the TXTLIB named TESTLIB as follows:

```
txtlib gen testlib myprog (filename
```

TESTLIB contains a member name MYPROG with an entry point BEGIN. Specify the member name MYPROG to reference this TXTLIB member. If you do not specify the FILENAME option, TESTLIB will contain no entry for the name MYPROG. You will have to specify the member name BEGIN to reference this TXTLIB member.

Loading and Executing TXTLIBs: When you want to load members of TXTLIBs into storage to execute them (just as you execute TEXT files), you must issue the GLOBAL command to identify the TXTLIB:

```
global txtlib testlib
load begin (start
```

When you specify more than one TXTLIB on the GLOBAL command line, the order of search is established for the TXTLIBs. However, if the AUTO option of the LOAD and INCLUDE commands is in effect (it is the default), CMS searches for TEXT files before searching active TXTLIBs.

When the TXTLIB command processes a TEXT file, it writes an LDT (loader terminate) card at the end of the TEXT file so that when a load request is issued for a TXTLIB member loading terminates at the end of the member. If you add OS linkage editor control statements to the TEXT file (using the CMS editor) before you issue the TXTLIB command to add the file to a TXTLIB, the control statements are processed as follows:

NAME Statement: A NAME statement causes the TXTLIB command to create the directory entry for the member using the specified name. Thereafter, when you want to load that member into storage or delete it from the TXTLIB you must refer to it by the name specified on the NAME statement.

Note: The FILename option overrides any name card found in a text file. The name card functions as before, but the specified file name becomes the membername in the TXTLIB. The name card is the only entry within that membername of the TXTLIB.

The loader does not use name cards to resolve entry points. It is important that the name on the name card be the same as the name on the CSECT or entry card. This will ensure that the loader will find the correct text deck and loader tables (any external references) will be resolved with the entry point. If the names differ, the loader will load the text deck based on the name card (or file name). However, the loader tables will be set up according to entry or CSECT cards encountered during the load. Any external reference using the name from the name card will be resolved as zeros.

ENTRY Statement: If you use an ENTRY statement, the entry point you specify is validated and checked for a duplicate. If the entry point name is valid and there are no duplicates in the TEXT file, the entry name is written in the LDT card. Otherwise, an error message is issued. When this member is loaded, execution begins at the entry point specified.

ALIAS Name: An entry is created in the directory for the ALIAS name you specify. A maximum of 16 alias names can be used in a single text deck. You may load the single member and execute it by referring to the alias name, but you cannot use the alias name as the object of V-type address constant (VCON) because the address of the member cannot be resolved.

SETSSI Card: TXTLIB command information you specify on the SETSSI card is written in bytes 26 through 33 of the LDT card.

All other OS linkage editor control statements and commands are ignored by the TXTLIB command and written into the TXTLIB member. When you attempt to load the member, the CMS loader flags these cards as invalid. These cards may be added as history information to a module if you specify the HIST option on the LOAD or INCLUDE commands and then issue a subsequent GENMOD command.

Manipulating TXTLIB Members

The following CMS commands recognize TXTLIBs and have a MEMBER option:

- PUNCH
- PRINT
- TYPE

Use the CMS PUNCH command with the MEMBER option to punch a single member or all members of a TXTLIB. Use the CMS PRINT command with the MEMBER option to print a single member or all members of a TXTLIB. Use the CMS TYPE command with the MEMBER option to display on the terminal a single member or all members of a TXTLIB.

OS Module Libraries and CMS LOADLIBS

The OS relocating loader allows the user to load a member of a CMS LOADLIB or an OS module library on an OS formatted disk. The OS LINK, LOAD, ATTACH, and XCTL macros are supported. In addition, the OSRUN command (which generates a LINK SVC) loads and executes members directly from the console.

For the LINK, LOAD, ATTACH, and XCTL macros, the libraries specified in the LOADLIB global list are searched. If the requested member is not found, CMS looks for a TEXT file by that name. Then, if still not found, the TXTLIBs specified in the TXTLIB global list are searched for the member name.

For the OSRUN command, the libraries specified in the LOADLIB global list are searched. If the member is not found and the user has a $SYSLIB LOADLIB file, it is searched for the member name. (TEXT files and TXTLIBs are not considered by OSRUN.)

Executing OS Module Libraries

If the module to be executed resides in an OS module library on an OS formatted disk, the disk must be accessed and the library must be defined (via the FILEDEF command) to make it known to CMS.

For example, access the OS disk as a B-disk at the address 250:

```
ACCESS 250 B
```

Suppose SYS1.TESTLIB is an OS module library on the OS disk and contains the member TEST1. Use the FILEDEF command to relate SYS1.TESTLIB to the CMS LOADLIB called OSLIB LOADIB:

```
FILEDEF $SYSLIB DISK OSLIB LOADLIB B DSN SYS1 TESTLIB
     (DSORG PO RECFM U BLOCK 7294
```

Now you can refer to the OS module library, SYS1.TESTLIB, by using the CMS file identifier OSLIB LOADLIB.

Before you try to execute TEST1, use the GLOBAL command to identify the CMS LOADLIBs to be searched. For example,

```
GLOBAL LOADLIB OSLIB
```

Then, the OSRUN command searches OSLIB LOADLIB for the member, TEST1, to load and execute. For example,

```
OSRUN TEST1
```

The DDNAME specified on the FILEDEF command must be $SYSLIB. The filename (OSLIB), specified on the FILEDEF command, can be any name, but it must correspond to the name stated in the GLOBAL command. The filetype must be LOADLIB.

Creating and Executing CMS LOADLIBs

If the program to be executed resides on a CMS disk, use the LKED command. The LKED command creates a CMS LOADLIB from a CMS TEXT file. For example:

```
LKED TESTFILE
```

takes the CMS TEXT file, TESTFILE TEXT, and creates the CMS LOADLIB, TESTFILE LOADLIB. For more information on input to the LKED command refer to "The LKED Command" on page 186. The CMS LOADLIB created by the LKED command is an OS simulated partitioned data set (PDS) named TESTFILE LOADLIB and contains one member named TESTFILE.

Before executing TESTFILE, use the GLOBAL command to identify the LOADLIB to be searched:

```
GLOBAL LOADLIB TESTFILE
```

Then the OSRUN command loads, relocates, and executes the TESTFILE member of TESTFILE LOADLIB:

```
OSRUN TESTFILE
```

Maintaining CMS LOADLIBs

The LOADLIB command provides the utility necessary to maintain the CMS LOADLIBs. The following functions are provided:

COPY	Copy members from one LOADLIB to another Merge complete LOADLIBs Copy with SELECT or EXCLUDE
COMPRESS	Compress a CMS LOADLIB
LIST	LIST members of a CMS LOADLIB

For more detailed information on the LKED, GLOBAL, OSRUN, and LOADLIB commands, refer to the *VM/SP CMS Command Reference.*

Concatenating Files

To define more than one library with the same DDNAME, use the CONCAT option of the FILEDEF command. You can concatenate the LOADLIB files on OS disks with each other and/or with CMS LOADLIB files. Any library to be searched must be specified in the GLOBAL LOADLIB statement. The data set with the largest block size should be specified first (both in the FILEDEF and in the GLOBAL list). CMS files do not require a file definition. But, if used, the file with the largest block size should be specified first. The GLOBAL list determines the order in which the libraries are searched.

For example, search two OS files and a CMS LOADLIB for the member, THETA, using the following commands:

```
ACCESS 250 B (if 250 is the address of the OS disk)
FILEDEF $SYSLIB DISK OSLIB LOADLIB DSN SYS1 LIB1
     (DSORG PO RECFM U BLOCK 7294)
FILEDEF $SYSLIB DISK MYLIB LOADLIB B DSN SYS1 LIB2 (CONCAT)
GLOBAL LOADLIB OSLIB MYLIB CMSLIB
OSRUN THETA
```

Note: The first FILEDEF command for $SYSLIB must describe the first library filename in the GLOBAL list. Its attribute will be used when the libraries are searched. It is advisable not to code the CONCAT option on the first FILEDEF command so that it clears all previous FILEDEFs for that ddname.

The LKED Command

The LKED command uses the OS Linkage Editor for the actual link of the TEXT file to the LOADLIB as an executable module. In order to link edit CMS files, you can issue the FILEDEF command to identify input to the OS Linkage Editor. Primary LKED input is a data set known to the linkage editor as SYSLIN, which can be described in the FNAME operand of the LKED command. The filetype of the input file named in the command line must be TEXT. Optionally, you can override the FNAME operand by issuing a FILEDEF that defines SYSLIN as the ddname of an alternate primary input source. If your alternate input is a CMS file, the choice of filetype is unrestricted. The contents of the SYSLIN dataset may be:

1. Object text such as assembler or compiler output
2. Linkage editor control statements
3. A combination of object text and control statements.

Linkage editor control statements can be inserted before, between, and after object modules and other control statements. Editing procedures can be used to construct files to meet your requirements. Linkage editor INCLUDE statements may be used to designate explicitly the following files or file members as secondary linkage editor input:

1. CMS TEXT files
2. Members of CMS TXTLIB files
3. Members of CMS LOADLIB files
4. Members of OS object libraries
5. Members of OS load libraries.

A FILEDEF must be issued before the LKED command to define a unique ddname for each file to be included as secondary linkage editor input. An INCLUDE statement in the SYSLIN dataset must specify the ddname assigned to the file by your FILEDEF. For library files, the statement must also specify all members of the library that are to be included as input. The use of all FILEDEF commands and INCLUDE statements to identify input files is shown in the following examples.

CMS commands:

```
FILEDEF LIBDEF DISK MYLIB TXTLIB B
FILEDEF TXTDEF DISK MYFILE TEXT C
```

SYSLIN input:

```
 INCLUDE LIBDEF(CSECT1,CSECT2)
 INCLUDE TXTDEF
```

INCLUDE statements must begin in column 2. The applicable statement formats are described in the *OS/VS Linkage Editor and Loader.*

When SYSLIN input to the LKED command is an assembled object file in fixed-block format residing on an OS disk, the RECFM FBS option of the FILEDEF command must be specified. The following example shows

FILEDEF commands and SYSLIN input to identify a member of an OS object library and a CMS TXTLIB.

CMS commands:

```
FILEDEF OSOBJ DISK OBJECT FILE Q DSN SYS1 FEOBJ (RECFM FBS
 LRECL 80 BLOCK 3120
FILEDEF TXTDEF DISK NEWLIB TXTLIB B
```

SYSLIN input:

```
INCLUDE OSOBJ(MEMBER1)
INCLUDE TXTDEF(CSECT1)
```

Automatic library search is available for either CMS or OS type library members if the FILEDEF for the dataset to be searched specifies SYSLIB as the ddname. Additional libraries can be selected for automatic search by placing linkage editor LIBRARY statements in your SYSLIN input file. Each library statement must contain the associated ddname and a list of members within the library to be included in the search. A FILEDEF must be issued before the LKED command to assign a unique ddname to each dataset to be searched. The library search conducted during a single linkage editor execution is limited to either object-type or load-type modules and may not combine both types. The CONCAT option of the FILEDEF command is not valid for LKED input datasets. To expand the use of the automatic SYSLIB search, the user may combine the members of several CMS libraries into a single composite library. The automatic search facility applies to CMS TXTLIBs and LOADLIBs and to OS object libraries and LOAD libraries. The following example shows FILEDEF commands and SYSLIN input for an automatic library search.

CMS commands:

```
FILEDEF SYSLIB DISK SEARCH1 TXTLIB B
FILEDEF LIBDEFA DISK SEARCH2 TXTLIB C
FILEDEF LIBDEFB DISK OSTEXT LIBRARY D DSN OBJMODS
```

SYSLIN input:

```
LIBRARY LIBDEFA(CSECT1,CSECT2)
LIBRARY LIBDEFB(MEMBER1,MEMBER2)
```

LIBRARY statements must begin in column 2. The GLOBAL command is not needed to identify linkage editor input libraries. For LOADLIB input to the linkage editor, the RECFM U option of the FILEDEF command must be specified.

The default FILEDEF commands issued by the LKED command for the ddnames presented to the Linkage Editor are as follows:

```
FILEDEF SYSLIN DISK FNAME TEXT * (RECFM F BLOCK 80 NOCHANGE
FILEDEF SYSLMOD DISK fname LOADLIB A1 (RECFM U BLOCK 260 NOCHANGE
-or-
FILEDEF SYSLMOD DISK libname LOADLIB A1 (RECFM U BLOCK 260 NOCHANGE
FILEDEF SYSUT1 DISK fname SYSUT1 *
FILEDEF SYSPRINT DISK fname LKEDIT A1
-or-
FILEDEF SYSPRINT PRINTER
-or-
FILEDEF SYSPRINT DUMMY
```

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

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 one that has been formatted by an OS program, such as the 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. See the section "CMS Support for OS and VSE/VSAM Functions" for details.

Refer to Figure 22 on page 189 and "OS Macros" on page 191, then read "Access Method Support" on page 199 to see how CMS handles these access methods.

Since CMS does not simulate the indexed sequential access method (ISAM), no OS program using 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 Disks

By simulating OS macros, CMS can read, but not write or update, OS sequential and partitioned data sets that reside on OS disks. 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 an OS system.

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.

Simulating OS Supervisor Calls

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
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

Figure 22 (Part 1 of 3). Simulated OS Supervisor Calls

Macro Name	SVC Number	Function
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
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

Figure 22 (Part 2 of 3). Simulated OS Supervisor Calls

Macro Name	SVC Number	Function
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 22 (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 multitasking 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 multitasking 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 Macros and Functions Reference* for descriptions of all CMS user macros.

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	2314
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.

If an STIMER is issued and later the virtual machine is put into a wait state, the virtual timer is not updated unless prior to this the CP command SET TIMER REAL was issued or the REALTIMER option was specified in the VM/SP directory entry of the virtual machine.

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[5]	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) 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, 1024, 2048, or 4096 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.

[5] 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.
• 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 the section "CMS Support for OS and VSE/VSAM Functions" for details.

CMS also updates those portions of the OS control blocks 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, 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. If the DCBRECFM is FB, the filemode number is 4, and the last block is a short block, 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.

BDAM Restrictions

The four methods of accessing BDAM records are:

1. Relative Block <u>RRR</u>
2. Relative Track <u>TTR</u>
3. Relative Track and Key <u>TTK</u>
4. Actual Address MBBCC<u>HHR</u>

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

BLDL	ENQ	RDJFCB
BSP	FIND	READ

CHECK	GET	SYNADAF
CLOSE	NOTE	SYNADRLS
DEQ	POINT	WAIT
DEVTYPE	POST	

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. If the data set 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.

OS Tape Volume Switching

DMSTVS is a CMS routine that performs tape volume switching operations for OS multivolume tape support. All tape volume switch requests are processed by this routine.

The DMSTVS function can be overridden by a nucleus extension with the name DMSTVS. The nucleus extension must follow the same entry and exit conventions as DMSTVS.

DMSTVS uses the TVSPARMS macro to set various values used by DMSTVS. These values include:

- The userid of the virtual machine that all CMS tape volume switching messages are sent.
- The time interval DMSTVS waits between issuing sense commands to determine the tapedrive's ready/notready status.
- The number of sense operations processed before issuing an additional tape volume switching prompt message.
- The number of prompt messages issued before message DMSTVS270I is issued.
- The number of prompt messages issued before message DMSTVS271I is issued.

- The decision to check for a write ring when NORING is requested.

Please note that DMSTVS sets the CP timer to real (CP SET TIMER REAL) and uses the OS STIMER simulation. This paces the prompting messages.

Chapter 9. Developing VSE Programs under CMS

You can use CMS to create, compile, test, execute, and debug VSE programs written in the Assembler, DOS/VS COBOL, DOS PL/I, DOS/VS RPG-II programming languages. CMS simulates many functions of the Disk Operating System VSE so you can use the interactive facilities of VM/SP to development and execute your VSE programs in a VSE virtual machine or in a batch facility virtual machine.

This chapter discusses the following topics:

- Entering the CMS/DOS environment
- Using DOS files on DOS disks
- Using the ASSGN command
- Using the DLBL command
- Using DOS libraries in CMS/DOS
- Using macro libraries
- VSE assembler language macros supported
- Assembling source programs
- Link-editing and executing programs in CMS/DOS
- VSE supervisor and I/O macros supported by CMS/DOS
- VSE supervisor control blocks simulated by CMS/DOS
- CMS/DOS user considerations and responsibilities

CMS/DOS is neither CMS nor is it DOS. It is a composite, and its vocabulary contains both CMS and VSE terms. CMS/DOS performs many of the same functions as DOS. However, under VSE a function is initiated by a control card, while under CMS it is initiated by a command. Many CMS/DOS commands, therefore, have the same names as the VSE control statement that performs the same function. In those cases where the control statement you would use in VSE and the command you use in CMS are different, the differences are explained. For the most part, whenever a term that is familiar to you as a VSE term is used, it has the same meaning to CMS/DOS, unless otherwise indicated.

CMS/DOS support in VM/SP is based on the VSE Program Product. The term DOS, however, continues to be used in a general sense, and in the discussion that follows, DOS refers to the VSE Program Product.

Note: VM/SP supports Version 1 Releases 2 and 3 of VSE/AF.

Entering the CMS/DOS Environment

After you have loaded CMS into your virtual machine, you can enter the CMS/DOS environment by issuing:

```
set dos on
```

If you want to access a DOS system residence volume during your CMS/DOS terminal session, you should link to and access the disk that contains the DOS SYSRES before you issue the SET command. For example, if you share the system residence volume with other users and it is in your directory at virtual address 390, you would issue the command:

```
access 390 g
```

then issue the SET command as follows:

```
set dos on g
```

to indicate that the SYSRES is located on your G-disk. If you are going to use the CMS/DOS librarian facilities to access any of the libraries on the system residence volume, you must enter the CMS/DOS environment this way.

If you are using CMS exclusively for DOS applications, you could put the ACCESS and SET DOS ON commands in your PROFILE EXEC.

All of the CP and CMS on-line debugging and testing facilities (such as the CP PER and STORE commands) are supported in the CMS/DOS environment. Also, CP disk error recording and recovery are supported in CMS/DOS.

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. If you are going to use access method services functions in CMS/DOS or execute functions that read or write VSAM data sets, you must use the VSAM option of the SET DOS ON command:

```
set dos on g (vsam
```

When you are using CMS/DOS, you can use your virtual machine just as you would if you were in the CMS environment. In the CMS/DOS environment, CMS supports many VSE facilities, but does not support OS simulation. For example, the SCRIPT command uses OS macros and is therefore invalid in the CMS/DOS environment. When you no longer need VSE support under CMS, you issue the SET DOS OFF command and VSE facilities are no longer available.

You have, however, in addition to the CP and CMS commands available, a number of CMS/DOS commands and CMS commands with special CMS/DOS operands that simulate VSE functions. Except for the DLBL and DOSLIB commands, these commands or operands should only be issued in the CMS/DOS environment.

The CMS/DOS commands and CMS commands with special CMS/DOS operands are summarized in Figure 23 on page 209. A detailed description of the commands and the command format are found in the *VM/SP CMS Command Reference.*

Command	Operand	Comments
ASSGN		Executable only in the CMS/DOS environment. Assigns CMS/DOS system or programmer logical units to a virtual device.
DLBL		Defines a VSE or VSAM ddname and relates the ddname to a disk file.
DOSLIB		Deletes, compacts, or lists information about the phases in a CMS/DOS phase library.
DOSLKED		Executable only in the CMS/DOS environment. Link-edits CMS text file or object modules from a VSE relocatable library, and places them in executable forms in a CMS/DOS phase library.
DOSPLI		Executable only in the CMS/DOS environment. Compiles DOS PL/I source programs.
DSERV		Executable only in the CMS/DOS environment. Displays information about VSE core image, relocatable, source statement, and procedure and/or transient directories.
ESERV		Executable only in the CMS/DOS environment. Displays, updates, punches, or prints edited (E sublibrary) VSE source statement books.
FCOBOL		Executable only in the CMS/DOS environment. Compiles DOS/VS COBOL source programs.
FETCH		Executable only in the CMS/DOS environment. Fetches a CMS/DOS executable phase.
GENMOD	OS DOS ALL	Specifies the type of macro support needed to execute a module. The ALL operand is intended for CMS internal use.
GLOBAL	DOSLIB	The GLOBAL command can specify CMS/DOS phase libraries, as well as text and macro libraries.
LISTIO		Executable only in the CMS/DOS environment. Display information about the CMS/DOS system and programmer logical units.
LOADMOD		Checks that a module generated to execute in a specific macro simulation environment (CMS/DOS or CMS) is in the correct environment.
OPTION		Executable only in the CMS/DOS environment. Sets compiler options for DOS/VS COBOL.
PSERV		Executable only in the CMS/DOS environment. Copies and displays procedures in the VSE procedure libraries and/or spools the procedures to the CMS virtual printer and/or punch.

Figure 23 (Part 1 of 2). CMS/DOS Commands and CMS Commands with Special Operands

Command	Operand	Comments
QUERY	UPSI	Executable only in the CMS/DOS environment.
		Displays current setting of CMS/DOS UPSI byte.
	OPTION	Displays the current data set definitions.
	DLBL	Executable only in the CMS/DOS environment.
		Displays CMS/DOS compiler options.
	DOSLNCNT	Displays the current number of SYSLST lines per page.
	DOS	Displays the current status (active or not active) of
	DOSLIB	CMS/DOS.
		Displays the names of all CMS/DOS phase libraries
	DOSPART	currently being searched for executable phases.
	LIBRARY	Displays the virtual partition size.
		Displays the names of all CMS/DOS phase libraries to be searched, in addition to the text and macro libraries.
RSERV		Executable only in the CMS/DOS environment. Copies and/or displays modules in a VSE relocatable library. Output can also be directed to the virtual printer or punch.
SET	DOS	Makes the CMS/DOS environment active or not active.
	DOSLNCNT	Specifies the number of SYSLST lines per page.
	nn	Sets the virtual partition size.
	DOSPART	Executable only in the CMS/DOS environment. Sets
	UPSI	the CMS/DOS UPSI byte.
SSERV		Executable only in the CMS/DOS environment. Copies or displays books from the VSE source statement library. Output can also be directed to the virtual printer or punch.

Figure 23 (Part 2 of 2). CMS/DOS Commands and CMS Commands with Special Operands

DL/I in the CMS/DOS Environment

Batch DL/I programs can be written and tested in the CMS/DOS environment. This includes programs written in assembler, COBOL, and PL/I languages. Not all functions of COBOL and PL/I are supported. For a description of what is supported, see the documentation on the appropriate licensed program.

Data base description generation and program specification block generation can also be executed. However, the application control block generation must be submitted to a DOS virtual machine for execution. The data base recovery and reorganization utilities must also be executed in a DOS virtual machine. This support provides the ability to:

- Interactively code DL/I control blocks and application programs that contain imbedded DL/I calls.

- Store and maintain macros used to generate DL/I control blocks in a CMS library. Store and maintain programs created under CMS in a CMS library. Production libraries are thus isolated from the test environment.

- Modify and compile programs using the CMS/DOS text manipulation and EXEC facilities.

- Link-edit and execute batch DL/I programs either interactively or in CMSBATCH. Online DL/I application programs requiring access to CICS/VS must be submitted to a DOS virtual machine for link-editing, cataloging, and execution.

The following restrictions apply:

- All the existing guidelines and restrictions that apply to VSAM data set creation, maintenance, and application program use apply to DL/I data sets.

- The CMS/DOS restriction on writing to sequential files applies to SHSAM and HSAM.

- To assemble a DBD or PSB under CMS/DOS, you must first copy the DBDGEN and PSBGEN macros from the DOS source statement library to a CMS MACLIB.

For more information about using DL/I in the CMS/DOS environment, see *DL/I DOS/VS Data Base Administration.*

Using DOS Files on DOS Disks

You can have DOS disks attached to your virtual machine by a directory entry or you can link to a DOS disk with the LINK command. You can use the ACCESS command to assign a mode letter to the disk:

```
access 155 b
```

and the RELEASE command to release it:

```
release b
```

Except for VSAM disks, you cannot write on DOS disks or update DOS files on them. You can, however, execute programs and CMS/DOS commands that read from these files, and you can use the LISTDS command to display the fileids of files on a DOS disk.

For example, if you enter:

```
listds b
```

You receive the following response, if the data set exists:

```
FM DATA SET NAME
B  NEW.TEST DATA
B  ONE.TEST ONE
B  TWO.TEST TWO
```

You can also verify the existence of a particular file. For example, if the file-id is NEW.TEST.DATA you can enter:

```
listds new.test.data.b
      -- or --
listds new test data b
```

If the file-id of the DOS file you want to verify contains embedded blanks, for example NEW.TEST DATA, then you have to enter the LISTDS commands with a question mark:

```
listds ? b
```

CMS responds:

```
Enter data set name:
```

and you can enter the exact file-id:

```
new.test data
```

If the data set exists, you receive a response:

```
FM DATA SET NAME
B  NEW.TEST DATA
```

Reading DOS Files

Under CMS/DOS, you can execute programs that read DOS sequential (SAM) files; you can also execute programs that read and write VSAM files. You cannot, however, execute programs to read direct (DAM) or indexed sequential (ISAM) DOS files. Complete information on using CMS to access and manipulate VSAM files is described in "Chapter 10. Using Access Method Services and VSAM under CMS and CMS/DOS" on page 273.

The discussion below lists the restrictions placed on reading SAM files.

CMS cannot read DOS files that:

- Have the input security indicator on.
- Contain more than 16 user labels and/or data extents. (If the file has user labels, they occupy the first extent. Therefore, the file must contain no more than 15 data extents.) User labels in user-labeled files are ignored.
- Are multivolume files. Multivolume files are read as single-volume files. End of volume is treated as end of file. There is no end-of-volume switching.

CMS does not support duplicate volume labels. You cannot access more than one volume with the same six-character label while you are using CMS/DOS.

Creating CMS Files from DOS Libraries

You can create CMS files from existing DOS files on DOS disks. CMS simulates the DOS librarian functions DSERV, RSERV, SSERV, ESERV, and PSERV with commands of the same names. You can use these CMS/DOS commands to create CMS files from relocatable source statement or procedure libraries located either on the DOS system residence volume or in private libraries. The functions are fully described later in this section.

Copying DOS Files and Tape Data Files

If you want to create CMS files from DOS files that are not cataloged in libraries or from DOS files on tape, you can use the MOVEFILE command. The MOVEFILE command allows you to copy a file from one device to another device of the same or a different type. Before issuing the MOVEFILE command, the input and the output files must be described to CMS with the FILEDEF command.

The MOVEFILE and FILEDEF commands are described in "Chapter 8. Developing OS Programs under CMS" on page 157. The procedures are the same for copying DOS files as for OS data sets. You must, however, keep the following in mind:

- Because DOS files on DOS disks do not contain BLKSIZE, RECFM, or LRECL options, these options must be specified via the FILEDEF command. Otherwise, default values are assigned. The default values are BLOCKSIZE=32760 and RECFM=U. LRECL is not used for RECFM=U files.

- If a DOS file-id does not follow OS naming conventions, you must use the DSN ? operand of FILEDEF and the ? operand of LISTDS to enter the DOS file-id. The OS naming conventions are: one-byte to eight-byte qualifiers, each qualifier must be separated by a period, and up to 44 characters including periods.

Copying Modules from VSE Library or SYSIN Tapes

You can create individual CMS files for VSE modules from a VSE library distribution tape or VSE SYSIN tape. Use the VMFDOS command. The VMFDOS command can create a CMS file for each VSE module that exists, and the CMS filename corresponds to the VSE module name. You can restore individual modules, groups of modules, or the entire module set.

For VSE library distribution tapes, the VMFDOS command restores modules from either system or private (relocatable and/or source statement) libraries. The created CMS files have a filetype of "TEXT" if they are from a relocatable library. They have a filetype of "MACRO" if they are from a source statement library.

For VSE SYSIN tapes, modules containing a period as the second character (for example, "A.") of a VSE "CATALx" control statement have a filetype of "MACRO." All other files have a filetype of "TEXT".

The VMFDOS command is described in the *VM/SP Installation Guide.*

Reading in Real Card Decks

If you have DOS files or source programs on cards, you can create CMS files directly by having these cards read into the real system card reader. You direct the cards to your virtual machine by punching a CP ID card. For example, if your userid is HARMONY, then enter:

```
ID HARMONY
```

and place this card in front of your card deck. When the cards appear in your virtual card reader, you can read them onto your CMS A-disk with the READCARD command:

```
readcard dataproc assemble
```

You can use the editor to remove any DOS control cards that may be included in the deck.

Using Tapes in CMS/DOS

See the *VM/SP CMS User's Guide* for a description of CMS tape label processing for CMS/DOS tape files. The support for tape labels is only for files defined by a DTFMT macro. If you do not use this macro, CMS bypasses IBM standard labels on input tapes and writes a tape mark over any existing labels on an output tape. The CMS LABELDEF command is equivalent in CMS/DOS to the VSE TLBL control statement when standard tape label processing is used.

The ASSGN Command

The ASSGN command performs the same function for CMS/DOS as the ASSGN control statement in VSE. The ASSGN command in CMS/DOS assigns a system or programmer logical unit (SYSxxx) to a virtual I/O device. A logical unit is a symbolic name a program may use to refer to a real I/O device without knowing the device address.

If the device is a disk, you can use the DLBL command to establish a real file identification for a symbolic filename in a program. The DLBL command is described under "Using the DLBL Command." As in VSE, you are not allowed to assign the system residence volume via the ASSGN command.

In addition to using the ASSGN command to relate real I/O devices with symbolic units, you must use it in CMS/DOS to:

- Assign SYSIN or SYSIPT for the input source file for a language compiler when you use the DOSPLI or FCOBOL commands.

- Identify the disk, by mode letter, on which a private core image, relocatable, or source statement library resides.

- Assign SYSIN or SYSIPT to the CMS disk on which an ESERV file, containing control statements for the ESERV program, resides.

When you enter the ASSGN command, you must supply the logical unit and the device. For example:

```
assgn sys100 printer
```

assigns the logical unit SYS100 to the printer. When you want to make an assignment to a disk device, you must specify the mode letter where the disk is accessed. You must also access the disk before making an assignment to the disk. The command:

```
assgn sys010 b
```

assigns the logical unit SYS010 to your B-disk.

Assigning System Logical Units

The system logical units you can assign and the valid device types you can assign to the units in CMS/DOS are listed below.

Some VSE system logical units cannot be assigned to a VSE formatted FB-512 device. These units are listed below. An error message is issued and the command terminated if any of the unsupported system logical units are specified in the ASSGN command.

SYSIPT, SYSRDR, SYSIN

You can assign SYSIPT, SYSRDR, and SYSIN to disk (mode), TAPE, or READER. If you make an assignment to SYSIN, both SYSRDR and SYSIPT are also assigned the same device. SYSIPT, SYSRDR, and SYSIN cannot be assigned to a VSE formatted FB-512 device.

SYSLST

You can assign SYSLST, the system logical unit for listings, to disk (mode), PRINTER, or TAPE. An assignment to DOS FB-512 disks is not supported.

SYSLOG

You can assign SYSLOG, terminal or operator output or messages, to PRINTER or TERMINAL. CMS/DOS always assigns SYSLOG to TERMINAL by default, so you never have to make this assignment except when you want to alter it.

SYSPCH

You can assign SYSPCH, punched output (for example, text decks), to PUNCH, disk (mode), or TAPE. An assignment to DOS FB-512 disks is not supported.

SYSCLB, SYSRLB, SYSSLB

You can assign SYSCLB, SYSRLB, and SYSSLB to private core image, relocatable, and source statement libraries, respectively. The only valid assignment for these units is to disk (mode). If you want to reference private libraries with the DOSLKED, DSERV, ESERV, FETCH, SSERV, or RSERV commands, you must assign SYSCLB, SYSRLB, or SYSSLB to the disks where the libraries reside.

Compiler I/O Assignments

The compilers supported by CMS/DOS expect input/output to be assigned to the following devices:

- SYSIN/SYSIPT must be assigned to the device where the input source file resides. Valid device types are reader, tape, or disk.

  You must assign SYSIN/SYSIPT. If it is unassigned at compilation time, an error message is issued and the FCOBOL or DOSPLI command is terminated.

- The user should assign the following logical units to any of the indicated device types:

  - SYSPCH and SYSLST to tape, punch, disk, or IGN

    If SYSPCH or SYSLST are unassigned at compilation time, the FCOBOL or DOSPLI EXEC file directs output to the disk where SYSIN resides if SYSIN is assigned to a read/write CMS disk. Otherwise, output is directed to the CMS read/write disk with the most read/write space.

  - SYSLOG to terminal

    If SYSLOG is unassigned, it is assigned to the terminal.

  - SYS001, SYS002, and SYS006 to disk.

    If SYS001, SYS002, and SYS006 are unassigned, output is directed to the CMS disk with the most read/write space.

  - SYS003-SYS005 to tape or disk.

    If SYS003 through SYS005 are unassigned, output is directed to the CMS disk with the most read/write space.

The maximum number of work files is six for DOS/VS COBOL Compiler (FCOBOL) and two for DOS PL/I Optimizing Compiler (DOSPLI).

Manipulating Device Assignments

You can assign programmer logical units SYS000 through SYS241 with the ASSIGN command. This deviates from VSE where the number of programmer logical units varies according to the number of partitions. Besides assigning I/O devices, the ASSGN command can also negate a previous assignment:

```
assgn syspch ua
```

Also, for a given device, ASSGN can specify that no real I/O operation (NOP) is to be performed during the execution of a program:

```
assgn sys009 ign
```

When you release a disk from your virtual machine, any assignments made to that disk are unassigned.

Listing I/O Assignments

You can find out the current assignments for system and programmer logical units with the LISTIO command, which lists all the system or programmer logical units, even those that are unassigned.

To list only currently assigned units, enter:

```
listio a
```

To find out the current assignment of one specific unit, for example SYS100, enter:

```
listio sys100
```

When you use the STAT option, LISTIO lists, for disk devices, whether the disk is read-only or read/write. For example, if you enter

```
listio sys100
```

you may receive the reply:

```
SYS100 B   R/W
```

This reply indicates that SYS100 is assigned to the B-disk, which is a read/write disk.

With the EXEC option of the LISTIO command, you can create a disk file containing the list of assignments. The name of the file is $LISTIO EXEC. It contains two EXEC numeric variables, &1 and &2, for each unit listed. For example, if you enter the command:

```
listio sys081 (exec
```

the file $LISTIO EXEC may contain the record:

```
&1 &2 SYS081 PRINTER
```

You can cancel all current assignments by leaving the CMS/DOS environment and then re-entering it:

```
set dos off
set dos on
```

Virtual Machine Assignments

When you assign a physical device type to a system or programmer logical unit, CMS relates the device to your virtual machine configuration. You receive an error message if you try to assign a logical unit to a device not in your configuration. For example, if you use the ASSGN command to assign a logical unit to a disk file, you must specify the access mode letter of the disk. If the disk is not accessed, the ASSGN command fails.

For another example, if you issue:

```
assgn syspch punch
```

the punch specified is your own virtual machine card punch. The actual destination of punched output then depends on the spooling characteristics of the punch. If it is spooled to another user or to *, then no real cards are punched; virtual card images are placed in the virtual reader of the destination userid, which may be another virtual machine or your own.

CMS supports only one reader, one punch, and one printer. You cannot make any assignments for multiple output devices in CMS/DOS. When you make an assignment for a logical unit that has already been assigned, it replaces the current assignment.

The DLBL Command

The DLBL command performs the same functions for CMS/DOS as the DLBL control statement in VSE. Use the DLBL command to supply CMS/DOS with specific file identification information for a disk file that is going to be used for input or output. For any DLBL command you issue, you must previously have issued an ASSGN command for the disk, specifying a system or programmer logical unit. The basic relationship is:

```
assgn SYSxxx mode
dlbl filename mode DSN ? (SYSxxx
```

Both the SYSxxx and the mode values must match on the ASSGN and DLBL commands. The disk where the file resides must be accessed at the specified mode.

Note: In the CMS/DOS and CMS/VSAM environments, filemodes "R" and "T" cannot be used on the DLBL command. These filemodes cannot be used because in CMS/DOS and CMS/VSAM, "R" and "T" are used as abbreviations for reader and terminal.

The filename on the DLBL command line, called a ddname in CMS/DOS, corresponds to the symbolic name for a file in a program. If you want to reference a private DOS library, you must use one of the following filenames:

System Logical Unit	**Filename**
SYSCLB	IJSYSCL
SYSRLB	IJSYSRL
SYSSLB	IJSYSSL

Entering File Identifications

When you issue the DLBL command you must identify the file, by file-id (for a VSE file) or by file identifier (for a CMS file). The keywords DSN and CMS indicate whether it is a VSE file or a CMS file, respectively.

If the file is a VSE file residing on a DOS disk, you can enter the DLBL command in one of three ways. For example, for a file named TEST.FILE.INPUT you may enter either:

```
assgn sys101 d
dlbl infile d dsn test.file.input (sys101

     -- or --

dlbl infile d dsn test file input (sys101

     -- or --

assgn sys101 d
dlbl infile d dsn ? (sys101
```

For any VSE file with a file-id that contains embedded blanks, you must use the "DSN ?" form, shown in the third example above. When you enter the DLBL command with the ? operand, you are prompted to enter the DOS file-id:

```
Enter data set name:
```

Then you can enter the DOS file-id. For example,

```
test.file.input
```

When you issue a DLBL command for a CMS file, you enter the filename and filetype following the keyword CMS:

```
assgn sys102 a
dlbl outfile a cms new output (sys102
```

In this example, if SYS102 is defined as an output file for a program, the output is written to your CMS A-disk in a file named NEW OUTPUT.

You can, for convenience, use a CMS default file identifier. If you enter:

```
dlbl outfile a cms (sys102
```

then the output filetype defaults to that of the ddname and the filename to FILE. So, this output file is named FILE OUTFILE.

Clearing and Displaying File Definitions

You can clear a DLBL definition for a file by using the CLEAR operand of the DLBL command:

```
dlbl outfile clear
```

To clear all existing definitions, except those entered with the PERM option, you can enter:

```
dlbl * clear
```

This command is issued by the assembler and the language processors when they complete execution. Definitions entered with the PERM option must be individually cleared.

Whenever you use the HX Immediate command to halt the execution of a program, the DLBL definitions in effect are cleared, including those entered with the PERM option.

You can find out what definitions are currently in effect by issuing the DLBL command with no operands:

```
dlbl
```

or you can use the QUERY command with the DLBL operand.

Using DOS Libraries in CMS/DOS

CMS/DOS provides you with the capability of using various types of files from DOS system or private libraries. You can copy, punch, display at the terminal, or print:

- Books from system or private source statement libraries using the SSERV command. Books refer to macros and source programs in a source statement library.
- Relocatable modules from system or private relocatable libraries using the RSERV command.

- Procedures from the system procedure library using the PSERV command.

You can also:

- Copy and de-edit macros from system and private E sublibraries using the ESERV command.
- Access the directories of system or private libraries using the DSERV command.
- Link-edit relocatable modules from system or private relocatable libraries with the DOSLKED command.
- Read core image phases from system or private core image libraries into storage for execution using the FETCH command.

The SSERV Command

If you have cataloged source programs or copy files in the system source statement library and you want to use CMS to modify and test them, you can copy them into CMS files using the SSERV command. For example, suppose you want to copy a book named PROCESS from the A sublibrary on the system residence volume. The DOS system residence is in your virtual machine configuration at virtual address 350, and you have accessed it as your F-disk. First, to indicate to CMS/DOS that the system residence is on your F-disk, you enter:

```
set dos on f
```

then you can enter the SSERV command, specifying the sublibrary identification and the book name:

```
sserv a process
```

This creates, from the A sublibrary, a file named PROCESS COPY and places it on your A-disk. If the book contained assembler language source statements you would want the filetype to be ASSEMBLE, so you may enter:

```
sserv a process assemble
```

If you want to copy a book from a private source statement library, you must first use the ASSGN and DLBL commands to make the library known to CMS/DOS. For example, to obtain a copy file from a private library on a DOS disk accessed as your D-disk, enter:

```
assgn syss1b d
dlbl ijsyssl d dsn ? (syss1b
Enter data set name:
program.test library
```

Now, when you enter the SSERV command:

```
sserv t setup copy
```

the book named SETUP in the T sublibrary of PROGRAM.TEST LIBRARY is copied into a CMS file named SETUP COPY. If SETUP is not found in the private library, then CMS searches the system library, if it is available.

The RSERV Command

In CMS/DOS, to manipulate relocatable modules that have been cataloged either on the system or a private relocatable library you must first copy them into CMS files with the RSERV command. You can link-edit modules directly from DOS relocatable libraries, but if you want to add or modify linkage editor control statements for a module, you must place the control statements in a CMS file.

If you are copying a relocatable module from the system relocatable library, you should make sure that you have indicated the system residence disk when you entered the CMS/DOS environment:

```
set dos on f
```

then you can issue the RSERV command specifying the name of the relocatable module you want to copy:

```
rserv rtna
```

The execution of this command results in the creation of a CMS file named RTNA TEXT on your A-disk.

If you want to copy a relocatable module from a private relocatable library, you must first use the ASSGN and DLBL commands to make the private library known to CMS/DOS:

```
assgn sysrlb d
dlbl ijsysrl d dsn reloc.lib (sysrlb
```

Then, issue the RSERV command for a specific module in that library:

```
rserv testrtna
```

to create the CMS file TESTRTNA TEXT from the module named TESTRTNA. If the module TESTRTNA is not found in RELOC.LIB, CMS searches the system library, if it is available.

The PSERV Command

If you want to copy DOS cataloged procedures into CMS files to use in preparing job streams for a DOS virtual machine, you can use the PSERV command:

```
pserv prepjob
```

This command creates a CMS file on your A-disk. The file is named PREPJOB PROC. To copy a procedure from the procedure library you must have entered the CMS/DOS environment specifying a disk mode for the system residence volume.

You cannot execute DOS/VSE procedures directly from the CMS/DOS environment. However, if you modify a procedure, you can punch it to a virtual machine that is running a DOS system and execute it there.

The ESERV Command

The CMS/DOS ESERV command is actually an EXEC procedure that calls the VSE ESERV utility program. To use the ESERV program, you first must IPL CMS with a CMSBAM DCSS (shared segment), then create a file with a filetype of ESERV that contains the ESERV control statements you want to execute.

For example, if you want to write a de-edited copy of the macro DTFCD onto your A-disk, you might create a file named DTFCD ESERV, with the record:

```
PUNCH E.DTFCD
```

Just as when you submit ESERV jobs in DOS, column 1 must be blank.

Prior to executing the ESERV program, you must enter the CMS/DOS environment by specifying the SET DOS ON command using a VSE system residence volume. This is necessary because the ESERV procedure invokes the ESERV program directly from the VSE core image library.

Then, you must assign SYSIN to the device on which the ESERV source file resides, usually your A-disk:

```
assgn sysin a
```

Then you can enter the ESERV command specifying the filename of the ESERV file:

```
eserv dtfcd
```

No other ASSGN commands are required. The CMS/DOS ESERV EXEC makes default assignments for SYSPCH and SYSLST to disk.

To copy and de-edit macros from a private E sublibrary, issue the ASSGN and DLBL commands to identify the library. For example, to identify a source statement library named TEST.MACROS on the DOS disk accessed as the C-disk, enter:

```
assgn sysslb c
dlbl ijsyssl c dsn test.macros (sysslb
```

The SYSLST output is contained in a CMS file with the same filename as the ESERV file and a filetype of LISTING. You must examine the LISTING file to see if the ESERV program executed successfully.

The SYSPCH output is contained in a file with the same name as the ESERV file and a filetype of MACRO. If you want to punch ESERV output to your virtual card punch, make an assignment of SYSPCH to PUNCH.

When you use the PUNCH or DSPCH ESERV control statements, CATAL.S, END, or /* records may be inserted in the output file. When you use the MACLIB command to add the MACRO file to a CMS macro library, these statements are ignored.

See "Using Macro Libraries" on page 225 for information on creating and manipulating CMS macro libraries.

The DSERV Command

You can use the DSERV command to examine the contents of system or private libraries. If you do not specify any options with it, the DSERV command creates a disk file, named DSERV MAP, on your A-disk. You can use the PRINT or TERM options to specify that the directory list is either to be printed or displayed at your terminal. You can also use the SORT option to create a sorted list.

In order to examine a system directory, you must have entered the CMS/DOS environment specifying the mode letter of the DOS system residence:

```
set dos on f
```

If you want to examine the directory of a private source statement, core image, or relocatable library you must issue the ASSGN and DLBL commands establishing SYSSLB, SYSCLB, or SYSRLB before using the DSERV command.

For example, to display at your terminal a list of procedures cataloged on the system procedure library, you would issue:

```
dserv pd (sort term
```

If the directory you are examining is for a core image library, you can specify a particular phase name to ascertain the existence of the phase:

```
dserv cd phase $$bopen (term
```

To list the directory of a private source statement library, you would first issue the ASSGN and DLBL commands:

```
assgn sysslb b
dlbl ijsyssl b dsn test.source (sysslb
```

then enter the DSERV command:

```
dserv sd
```

The CMS file, DSERV MAP A, contains the directory of the private source statement library TEST.SOURCE.

The DOSLKED Command

You can use the DOSLKED command to link-edit relocatable modules from system or private relocatable libraries and to place these modules in a phase library (DOSLIB). CMS searches for a module in a private relocatable library before searching in a system relocatable library.

For more information on using the DOSLKED command, see "Link-editing Programs in CMS/DOS" on page 240.

DOS Core Image Libraries

You can load core image phases from DOS core image libraries into virtual storage and execute them under CMS/DOS. Since CMS cannot write directly to DOS disks, linkage editor output under CMS/DOS is placed in a special CMS file called a DOSLIB. When you execute the FETCH command in CMS/DOS you can load phases from either system or private DOS core image libraries as well as from CMS DOSLIBs. More information on using the FETCH command is contained under "Executing Programs in CMS/DOS" on page 243.

Using Macro Libraries

DOS macro libraries cannot be accessed directly by the VM/SP assembler. If you want to assemble DOS programs in CMS/DOS that use DOS macro or copy files that are on the system or a private macro library, you must first create a CMS macro library (MACLIB) containing the macros you wish to use. Since the process of creating a CMS MACLIB from the DOS toptem source statement library (E sublibrary) can be very time-consuming, you should check with your installation's system programmer to see if it has already been done and to verify the filename of the macro library, so that you can use it in CMS/DOS.

Note: The DOS PL/I and DOS/VS COBOL compilers executing in CMS/DOS cannot read macro or copy files from CMS MACLIBs. Macros and copy files are obtained instead from a DOS source statement library.

If you want to extract DOS system macros to modify them for your private use, or if you want to use macros from a private library in CMS, you must use the procedure outlined below to create the MACLIB files.

Creating CMS MACLIBs

A CMS macro library has a filetype of MACLIB. You can create a MACLIB from files with filetypes of MACRO or COPY. A MACRO file may contain macro definitions. COPY files contain predefined source statements.

To create a CMS macro library, each macro or copy file you want included in the MACLIB must first be contained in a CMS file with a filetype of COPY or MACRO. If you are creating a CMS MACLIB file from a DOS library, you must use the SSERV command to copy a file from any source statement library other than an E sublibrary or use the ESERV command to copy and de-edit a macro from an E sublibrary. The SSERV command uses a default filetype of COPY. The ESERV command uses a default filetype of MACRO.

The following example shows how to copy macros from various sources and shows how to create and use the CMS MACLIB that contains these macros:

1. Enter the CMS/DOS environment with the DOS system residence on a disk accessed as mode C:

   ```
   set dos on c
   ```

2. Copy the macro book named OPEN from the A sublibrary of the system source statement library:

   ```
   sserv a open
   ```

3. Establish a private source statement library:

   ```
   access 351 d
   assgn sysslb d
   dlbl ijsyssl d dsn ? (sysslb
   test source.lib
   ```

4. Issue the SSERV command for a macro in the M sublibrary of TEST SOURCE.LIB:

   ```
   sserv m releas
   ```

5. Create an ESERV file to copy from the E sublibrary:

   ```
   xedit contrl eserv
   input punch contrl
   file
   ```

6. Execute the ESERV command:

   ```
   assgn sysin a
   eserv contrl
   ```

7. Create a CMS macro library named MYDOSMAC from the files just created, which are named OPEN COPY, RELEAS COPY, and CONTRL MACRO:

   ```
   maclib gen mydosmac open releas contrl
   ```

   See "The MACLIB Command" on page 227 for more details.

8. To use these macros in an assembler language program, you must indicate that this MACLIB is accessible before assembling a source file:

```
global maclib mydosmac
```

See “The GLOBAL Command” on page 235 for more details.

Rather than issuing these commands every time you want to copy and create macros, you can put these commands in an EXEC.

The MACLIB Command

The MACLIB command performs a variety of functions. You use it to:

- Create the MACLIB (GEN function).
- Add, replace, or delete members (ADD, REP, and DEL functions).
- Compress the MACLIB (COMP function).
- List the contents of the MACLIB (MAP function).

Descriptions of these MACLIB command functions follow.

Creating a Macro Library

The GEN (generate) function creates a CMS macro library from input files specified on the command line. The input files must have filetypes of either MACRO or COPY. For example:

```
maclib gen mymac get pdump put regequ
```

creates a macro library with the file identifier MYMAC MACLIB A1 from macros existing in the files with the file identifiers:

GET $\left\{\begin{matrix}\texttt{MACRO}\\\texttt{COPY}\end{matrix}\right\}$,PDUMP $\left\{\begin{matrix}\texttt{MACRO}\\\texttt{COPY}\end{matrix}\right\}$,PUT $\left\{\begin{matrix}\texttt{MACRO}\\\texttt{COPY}\end{matrix}\right\}$,and REGEQU $\left\{\begin{matrix}\texttt{MACRO}\\\texttt{COPY}\end{matrix}\right\}$

If a file named MYMAC MACLIB A1 already exists, it is erased.

Assume that the files GET MACRO, PDUMP COPY, PUT MACRO, and REGEQU COPY exist and contain macros in the following form:

GET MACRO	PDUMP COPY	PUT MACRO	REGEQU COPY
GET	*COPY PDUMP	PUT	XREG
	PDUMP		
WAIT	*COPY WAIT		YREG
	WAIT		

The resulting file, MYMAC MACLIB A1, contains the members:

```
GET            WAIT
WAIT           PUT
PDUMP          REGEQU
```

The WAIT macro, which appears twice in the input to the command, also appears twice in the output. The MACLIB command does not check for duplicate macro names. If, at a later time, the WAIT macro is requested from MYMAC MACLIB, the first WAIT macro encountered in the directory is used.

When COPY files are added to MACLIBs, the name of the library member is taken from the name of the COPY file or from the *COPY statement, as in the file PDUMP COPY, above.

Note: Although the file REGEQU COPY contained two macros, they were both included in the MACLIB with the name REGEQU. When the input file is a MACRO file, the member name is taken from the macro prototype statement in the MACRO file.

Adding a Member to a Macro Library

The ADD function appends new members to an existing macro library. For example, assume that MYMAC MACLIB A1 exists as created in the example in the explanation of the GEN function and the file DTFDI COPY exists as follows:

```
*COPY DTFDI
 DTFDI macro definition
*COPY DIMOD
 DIMOD macro definition
```

If you issue the command:

```
maclib add mymac dtfdi
```

the resulting MYMAC MACLIB A1 contains the members:

```
GET             PUT
WAIT            REGEQU
PDUMP           DTFDI
WAIT            DIMOD
```

Replacing a Member of a Macro Library

The REP (replace) function deletes the directory entry for the macro definition in the files specified. It then appends new macro definitions to the macro library and creates new directory entries. For example, assume that a macro library TESTMAC MACLIB contains the members ALPHA, BETA, and SIGMA, and that the following command is entered:

```
maclib rep testmac alpha sigma
```

The files represented by file identifiers ALPHA MACRO and SIGMA MACRO each have one macro definition. After execution of the command, TESTMAC MACLIB contains members with the same names as before, but the contents of ALPHA and SIGMA are different.

Deleting a Member of a Macro Library

The DEL (delete) function removes the members from the macro library directory and compresses the directory so there are no unused entries. The macro definition still occupies space in the library, but since no directory entry exists, it cannot be accessed or retrieved. If you attempt to delete a macro for which two macro definitions exist in the macro library, only the first one encountered is deleted. For example:

```
maclib del mymac get put wait dtfdi
```

deletes macro names GET, PUT, WAIT, and DTFDI from the directory of the macro library named MYMAC MACLIB. Assume that MYMAC exists as in the ADD function example. After the above command, MYMAC MACLIB contains the following members:

```
PDUMP
WAIT
REGEQU
DIMOD
```

Compressing a Macro Library

Execution of a MACLIB command with the DEL or REP functions can leave unused space within a macro library. The COMP (compress) function removes any macros that do not have directory entries. This function uses a temporary file named MACLIB CMSUT1. For example, the command:

```
maclib comp mymac
```

compresses the library MYMAC MACLIB.

Listing Information about Members of a Macro Library

The MAP function creates a list containing the name of each macro in the directory, the size of the macro, and its position within the macro library. If you want to display a list of the members of a MACLIB at the terminal, enter the command:

```
maclib map mymac (term
```

The default option, DISK, creates a file on your A-disk which has a filetype of MAP and a filename equal to the filename of the MACLIB. If you specify the PRINT option, then a copy of the map file is spooled to your virtual printer as well as being written onto disk.

Note: The DISK, PRINT, and TERM options erase the old MAP file, if one exists.

You can also retrieve information for specific members of the library by indicating the member names following the MAP operand. For example:

```
maclib map mylib swerve yield
```

returns the information for only members SWERVE and YIELD of MYLIB MACLIB.

If you want to place that information in the program stack, use the STACK option of the MAP operand. The information can be stacked FIFO (first-in first-out) or LIFO (last-in first-out). The default order when STACK is specified alone is FIFO. The options STACK, STACK FIFO, and FIFO are equivalent. The options STACK LIFO and LIFO are equivalent. For example:

```
maclib map mylib neutral reverse (stack fifo
```

stacks in the program stack, the returned information for the NEUTRAL and REVERSE members of MYLIB in first-in first-out order.

Manipulating MACLIB Members

The following CMS commands have a MEMBER option, which allows you to reference individual members of a MACLIB:

- PRINT (to print a member)
- PUNCH (to punch a member)
- TYPE (to display a member)
- FILEDEF (to establish a file definition for a member)
- XEDIT (to create and/or edit a specific member).

You can use the editor to create the MACRO and COPY files and then use the MACLIB command to place them in a library. Once they are in a library, you can erase the original files, or you can edit a member of a CMS library using the XEDIT command with the MEMBER option. For example, entering the command:

```
xedit mylib maclib a1 (member swerve
```

If the SWERVE member does not exist in that library, a new file is created with a fileid of SWERVE MEMBER A1. If SWERVE is an existing member of MYLIB MACLIB, then you can edit the file.

You can also select members of a specific CMS library to edit from your MACLIST (invoked by the MACLIST command).

Note: You cannot create a new MACLIB using the MEMBER option of the XEDIT command. You must use the MACLIB command with the GEN option to create a new MACLIB.

To extract a member from a macro library, you can use either the PUNCH or the MOVEFILE command. If you use the PUNCH command, you can spool your virtual card punch to your own virtual reader:

```
cp spool punch to *
```

Then punch the member:

```
punch testmac maclib (member get noheader
```

and read it back onto disk:

```
readcard get macro
```

In the above example, the member was punched with the NOHEADER option of the PUNCH command, so that a name could be assigned on the READCARD command line. If a header had been created for the file, it would have indicated the filename and filetype as GET MEMBER.

If you use the MOVEFILE command, you must issue a file definition for the input member name and the output macro or copy file before entering the MOVEFILE command:

```
filedef inmove disk testcopy maclib (member enter
filedef outmove disk enter copy a
movefile
```

This example copies the member ENTER from the macro library TESTCOPY MACLIB A into a CMS file named ENTER COPY.

When you use the PUNCH or MOVEFILE commands to extract members from CMS MACLIBs, each member is followed by a // record, which is a MACLIB delimiter. You can edit the file and use the DELETE subcommand to delete the // record.

If you wish to move the complete MACLIB to another file, use the COPYFILE command.

To print a single member or all members of a macro library, use the CMS PRINT command with the MEMBER option. To display on the terminal a single member or all members of a macro library, use the CMS TYPE command with the MEMBER option.

The MACLIST Command

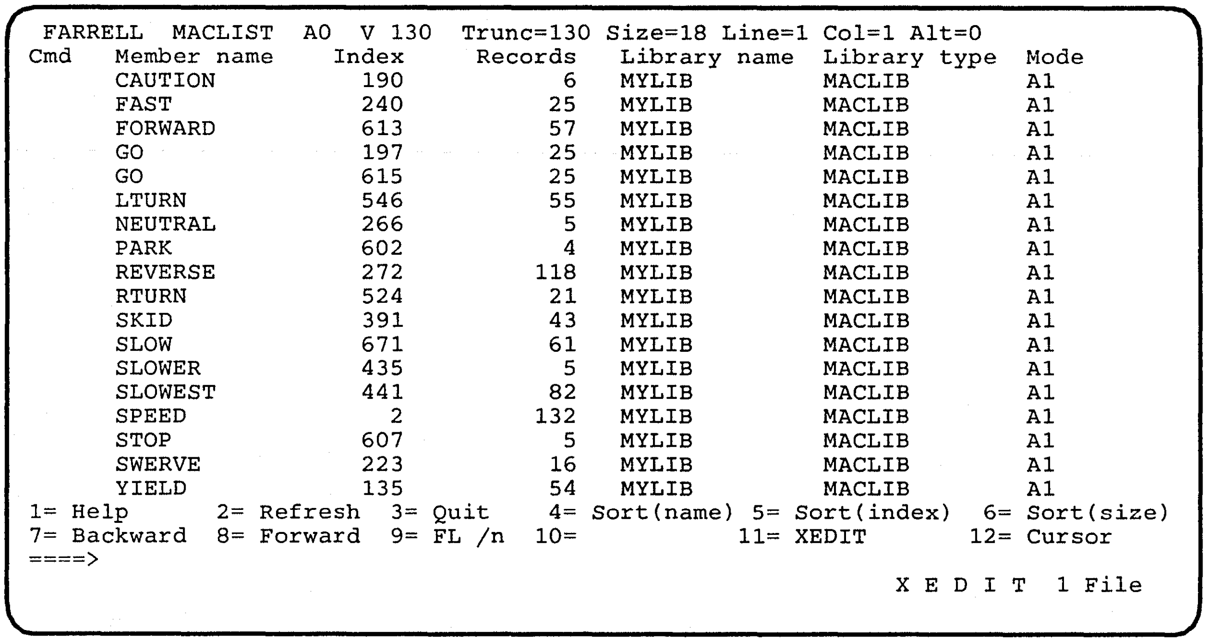
The MACLIST command displays a list of all members in a specified macro library. MACLIST provides you with an easy way to select and edit CMS maclib members. CMS commands can be issued against the members directly from the displayed list. The commands execute when you press the ENTER key (which is set to the EXECUTE command).

In the MACLIST environment, information that is normally provided by the MACLIB command (with the MAP option) is displayed under the control of the System Product editor. You can use XEDIT subcommands to manipulate the list itself.

The following MACLIST screen was created by issuing the MACLIST command as follows:

```
maclist mylib
```

Note that the members are sorted alphabetically by member name. Members with the same name are then sorted by index number (least to greatest).



```
 FARRELL  MACLIST  A0  V 130  Trunc=130 Size=18 Line=1 Col=1 Alt=0
Cmd    Member name    Index    Records   Library name  Library type  Mode
       CAUTION          190          6   MYLIB         MACLIB        A1
       FAST             240         25   MYLIB         MACLIB        A1
       FORWARD          613         57   MYLIB         MACLIB        A1
       GO               197         25   MYLIB         MACLIB        A1
       GO               615         25   MYLIB         MACLIB        A1
       LTURN            546         55   MYLIB         MACLIB        A1
       NEUTRAL          266          5   MYLIB         MACLIB        A1
       PARK             602          4   MYLIB         MACLIB        A1
       REVERSE          272        118   MYLIB         MACLIB        A1
       RTURN            524         21   MYLIB         MACLIB        A1
       SKID             391         43   MYLIB         MACLIB        A1
       SLOW             671         61   MYLIB         MACLIB        A1
       SLOWER           435          5   MYLIB         MACLIB        A1
       SLOWEST          441         82   MYLIB         MACLIB        A1
       SPEED              2        132   MYLIB         MACLIB        A1
       STOP             607          5   MYLIB         MACLIB        A1
       SWERVE           223         16   MYLIB         MACLIB        A1
       YIELD            135         54   MYLIB         MACLIB        A1
1= Help      2= Refresh  3= Quit    4= Sort(name) 5= Sort(index)  6= Sort(size)
7= Backward  8= Forward  9= FL /n  10=           11= XEDIT       12= Cursor
====>
                                                         X E D I T  1 File
```

Figure 24. Sample MACLIST Screen

Finding Members in Your MACLIST List

If there are many members in the maclib, the list may take up more than one screen. To find a member in your MACLIST list, you can do any of the following:

- Scroll through the list using the PF keys.

PF7	Scrolls backward one full screen.
PF8	Scrolls forward one full screen.

- Rearrange the list using one of the following PF keys:

PF4	Sorts the list by member name. This is how the list is initially arranged.
PF5	Sorts the list by index (largest first). The most recently updated members will have a greater number.
PF6	Sorts the list by size (largest to smallest).

- Use the XEDIT subcommand LOCATE if you know the member name that you are looking for.

- Rearrange the list by entering one of the following synonyms on the command line.

SINDEX	Sorts the list by index (greatest to least) within a library.
SLIB	Sorts the list alphabetically by library fileid.
SNAME	Sorts the list alphabetically by member name. This is how the list is initially arranged.
SSIZE	Sorts the list by member size (number of records, greatest to least).

Entering Commands in the MACLIST Environment

You can type commands that operate on member names in the list directly on the lines of the MACLIST display. When you press the ENTER key, all commands typed on the lines in the file displayed on the current screen are executed. Symbols can be used to represent operands in the command to be executed. Symbols are needed if the command to be executed has operands or options that follow the fileid. For example to issue the PRINT command for this member of your MACLIST:

```
      NEUTRAL        266         5   MYLIB     MACLIB      A1
```

type directly on the line that contains this member as follows:

```
print /EUTRAL        266         5   MYLIB     MACLIB      A1
```

and then press the ENTER key. Refer to the MACLIST command in the *VM/SP CMS Command Reference* for more information about using symbols in MACLIST.

Another way to issue commands that make use of member names displayed is to move the current line to the first (or only) member you want the command to use. Then issue EXECUTE (in the form “EXECUTE lines command”) from the XEDIT command line. This method may be used on both display and typewriter terminals. You can also enter commands from the MACLIST command line.

Editing a Maclib Member: The MACLIST command allows you to select and edit a CMS maclib member from the list. To edit a member, position the cursor on the line that contains the member to be edited and press the PF11 key. Otherwise, you can edit a CMS maclib member by using the XEDIT command with the MEMBER option. For example, to edit the SWERVE member of MYLIB maclib, enter:

```
xedit mylib maclib a1 (member swerve
```

If the SWERVE member did not exist in MYLIB MACLIB, a new file is created with a fileid of SWERVE MEMBER A1.

Adding and Replacing Maclib Members: When the MEMBER option is specified for the XEDIT command for a member that does not exist in the library, a new file is created with the fileid of "membername MEMBER fm".

If the MEMBER option is specified on the XEDIT command for an existing member of a library, the member is read into a file called "membername MEMBER fm" for you to edit.

When you issue FILE or SAVE for the new or changed member, the library directory is updated. The new or changed member and the updated library directory are added to the end of the library. If the directory already contains a member with the same name as the one being saved, the old entry is blanked out, so that the updated member replaces the old version.

Deleting Maclib Members: Use the DISCARD command to delete a member from a library. DISCARD is equivalent to the CMS command MACLIB DEL. DISCARD can either be typed in the command area of the line that describes the member you want discarded, or it can be entered from the command line (at the bottom of the screen). DISCARD can only be used while in the FILELIST, RDRLIST, MACLIST, and PEEK command environments.

Setting MACLIST Defaults: When XEDIT is invoked by the MACLIST command to display the list, the default XEDIT macro, PROFMLST XEDIT, is executed. If you want to invoke a different XEDIT macro, you can specify the PROFILE option with the MACLIST command. For example, to invoke MACLIST with the MYMCLST XEDIT macro, enter

```
maclist mylib (profile mymclst
```

You can do the same with the COMPACT and NOCOMPACT options of the MACLIST command.

If you are using an alternate profile most of the time, you may change the default profile with the DEFAULTS command. For example:

```
defaults set maclist profile mymclst
```

Entering the DEFAULTS command with no options provides you with the status of defaults currently in effect. For example, entering

```
defaults
```

after changing the XEDIT macro, returns the following information:

```
The following default options have been set:

Filelist options = PROFILE PROFFLST NOFILELIST
Help options = SCREEN BRIEF ALL
Maclist options = PROFILE MYMCLST NOCOMPACT
Note options = PROFILE PROFNOTE SHORT LOG NOACK NOTEBOOK ALL
Peek options = PROFILE PROFPEEK FROM 1 FOR 200
Rdrlist options = PROFILE PROFRLST
Receive options = LOG OLDDATE NOTEBOOK ALL
Sendfile options = NEW TYPE NOFILELIST LOG NOACK
Tell options = MSGCMD MSG

To change any default options enter DEFAULTS Set Cmdname Opt1 <Opt2..>
```

The GLOBAL Command

When you want to assemble a source program that uses macro or copy definitions, you must ensure that the library containing the code is identified before you invoke the assembler. Otherwise, the library is not searched. You identify libraries to be searched using the GLOBAL command. For example, if you have two MACLIBs that contain your private macros and copy files whose names are TESTMAC MACLIB and TESTCOPY MACLIB, you would issue the command:

```
global maclib testmac testcopy
```

The libraries you specify on a GLOBAL command line are searched in the order you specify them. A GLOBAL command remains in effect for the remainder of your terminal session, until you issue another GLOBAL MACLIB command or until you IPL CMS. To find out what macro libraries are currently available for searching, issue the command:

```
query maclib
```

You can reset the libraries or the search order by reissuing the GLOBAL command.

System MACLIBs

The macro libraries that are on the system disk contain CMS and OS assembler language macros you may want to use in your programs. The MACLIBs are:

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

  Note: When assembling programs that use CMS macros, both of these libraries should be identified via the GLOBAL command. DMSSP should precede CMSLIB in the search order.
- DOSMACRO MACLIB contains macros used internally by CMS/DOS.

Note: These macros should not be used in user written programs. To assemble programs that use VSE macros, you should follow the procedures as previously described in this chapter.

- OSMACRO MACLIB, OSMACRO1 MACLIB, and TSOMAC MACLIB are used by OS programmers.
- DMKSP MACLIB contains macros that support CP.
- OSVSAM MACLIB contains the subset of supported OS/VSAM macros.

To obtain a list of macros in any of these libraries, use either the MACLIST command or the MACLIB command with the MAP function. In the MACLIST environment, you can issue CMS commands against the members directly from the displayed list. You can find the MACLIST command description in the *VM/SP CMS Command Reference.*

When you use VSAM on CMS and write programs using VSE/VSAM macros, you can build a VSE/VSAM maclib by issuing the CMS VSEVSAM command. The maclib contains the supported VSE/VSAM macros and the following VSE macros:

```
CDLOAD
CLOSE
CLOSER
GET
OPEN
OPENR
PUT
```

Refer to the *VM/SP Installation Guide* for the CMS VSEVSAM command documentation.

VSE Assembler Language Macros Supported

Figure 25 on page 237 lists the VSE assembler language macros supported by CMS/DOS. You can assemble source programs that contain these macros under CMS/DOS, provided that you have the macros available in either your own or a shared CMS macro library. The macros whose functions are described in the "Function" column with the term "no-op" are supported for assembly only; when you execute programs that contain these macros, the VSE functions are not performed. To accomplish the macro function you must execute the program on a real VSE system.

Macro Name	SVC Number	Function
CALL		Pass control to another program
CANCEL	06	Terminate processing
CDLOAD	65	Load a VSAM phase
CHECK		Verify completion of a read or write operation
CLOSE/ CLOSER		Deactivate a data file
CNTRL		Control a physical device
COMRG	33	Return address of background partition communication region
DEQ	41	no-op
DTFxx		Establish file definitions
DUMP		Dump storage and registers and terminate processing
ENQ	42	no-op
EOJ	14	Terminate processing normally
ERET		Provide an error routine
EXCP	00	Execute a channel program
EXIT PC	17	Return from program check routine
EXIT AB	95	Return from abnormal termination routine
EXTRACT	98	Retrieve PUB, storage boundaries, or CPUID information
FCEPGOUT	86	no-op
FETCH	01	Load and pass control to a phase
	02	Load and pass control to a logical transient
FREE	36	no-op
FREEVIS	62	Release user free storage
GENL		Generate a phase directory list
GET		Access a sequential file
GETFLD/ MODFLD	107	Provide macro interface support for system information retrieval.
GETVCE	99	Return requested device information to output area.
GETVIS	61	Obtain user free storage
GETIME	34	Get the time of day
JDUMP		Dump storage and registers and terminate processing
LOAD	04	Load a phase into storage
LOCK/ UNLOCK	110	Resource control
MVCOM	05	Modify bytes in the partition communication region
NOTE		Manage data set access
OPEN/ OPENR		Activate a data file

Figure 25 (Part 1 of 2). VSE Macros Supported by CMS

Macro Name	SVC Number	Function
PAGEIN	87	no-op
PDUMP		Dump storage and registers and continue processing
PFIX	67	no-op
PFREE	68	no-op
POINTR		Position a file for reading
POINTS		Reposition a file to its beginning
POINTW		Position a file for writing
POST	40	Post the event control block
PRTOV		Control printer overflow
PUT		Write to a sequential file
PUTR		Communicate with the system operator
READ		Access a sequential file
RELPAG	85	no-op
RELSE		Skip to begin reading next block
RETURN		Return control to calling program
RUNMODE	66	Check if program is running real or virtual
SECTVAL	75	Obtain a sector number
SETIME	10/24	no-op
SETPFA	71	no-op
STXIT AB	37	Provide or terminate linkage to abnormal ending routine
STXIT PC	16	Provide or terminate linkage to program check routine
STXIT IT	20	no-op
STXIT OC	18	no-op
SUBSID	105	Retrieve information on supervisor subsystem
TRUNC		Skip to begin writing next block
TTIMER	52	Return a 0 in Register 0 (effectively a no-op)
WAIT	07	Wait for the completion of I/O
WRITE		Write to a sequential file
xxMOD		Create Logical IOCS routine inline

Figure 25 (Part 2 of 2). VSE Macros Supported by CMS

Assembling Source Programs

If you are a DOS assembler language programmer using CMS/DOS, you should be aware that the assembler used is the VM/SP assembler, not the DOS assembler. The major difference is that the VM/SP assembler, invoked by the ASSEMBLE command, is designed for interactive use. Therefore, when you assemble a program, error messages are displayed at your terminal when compilation is completed, and you do not have to wait for a printed listing to see the results. You can correct your source file and reassemble it immediately. Then you can print the listing error free.

To specify options to be used during the assembly, you enter them on the ASSEMBLE command line. So, for example, if you do not want the output LISTING file placed on disk, you can direct it to the printer:

```
assemble myfile (print
```

All of the ASSEMBLE command options are listed in *VM/SP CMS Command Reference.*

When you invoke the ASSEMBLE command specifying a file with a filetype of ASSEMBLE, CMS searches all of your accessed disks, using the standard search order, until it locates the file. When the assembler creates the output LISTING and TEXT files, it writes them onto disk according to the following priorities:

1. If the source file is on a read/write disk, the TEXT and LISTING files are written onto the same disk.

2. If the source file is on a read-only disk that is an extension of a read/write disk, the TEXT and LISTING files are written onto the parent disk.

3. If the source is on any other read-only disk, the TEXT and LISTING files are written onto the A-disk.

In all of the above cases, the filenames assigned to the TEXT and LISTING files are the same as the filename of the input file.

The output files used by the assembler are defined via FILEDEF commands issued by CMS when it calls the assembler. If you issue a FILEDEF command using one of the assembler ddnames before you issue the ASSEMBLE command, you can override the default file definitions.

The ddname for the source input file is ASSEMBLE. If you enter:

```
filedef assemble reader
assemble sample
```

then the assembler reads your input file from your card reader, and assigns the filename SAMPLE to the output TEXT and LISTING files. You can use this method to assemble programs directly from DOS sequential files on DOS disks. For example, to assemble a source file named DOSPROG from a DOS disk accessed as a C-disk, you could enter:

```
filedef assemble c dsn dosprog (recfm f lrecl 80
assemble dosprog
```

Again, the name you assign on the ASSEMBLE command may be anything. The assembler uses this name to assign filenames to the TEXT and LISTING output files.

LISTING and TEXT are the ddnames assigned to the SYSLST and SYSPCH output of the assembler. You might issue file definitions to override these defaults as follows:

```
filedef listing disk assemble listfile a
filedef text disk assemble textfile a
assemble source
```

When these commands are executed, the output from the assembly of the file SOURCE ASSEMBLE is written to the disk files ASSEMBLE LISTFILE and ASSEMBLE TEXTFILE.

Link-editing Programs in CMS/DOS

When the assembler or one of the language compilers executes, the object module produced is written to a CMS disk in a file with a filetype of TEXT. The filename is always the same as that of the input source file. These TEXT files (sometimes referred to as decks, although they are not real card decks) can be used as input to the linkage editor or can be used with an INCLUDE linkage editor control statement.

You can invoke the CMS/DOS linkage editor with the DOSLKED command, for example:

```
doslked test testlib
```

where TEST is the filename of either a DOSLNK or TEXT file (that is, a file with a filetype of either DOSLNK or TEXT) or the name of a relocatable module in a system or private relocatable library. TESTLIB indicates the name of the output file which, in CMS/DOS, is a phase library with a filetype of DOSLIB.

When you issue the DOSLKED command:

1. CMS first searches for a file with the specified name and a filetype of DOSLNK.

2. If none is found, CMS searches the private relocatable library, if you have assigned one. You must issue an ASSGN command for SYSRLB and use the ddname IJSYSRL in a DLBL statement.

3. If the module is still not found, CMS searches all of your accessed disks for a file with the specified name and a filetype of TEXT.

4. Last, CMS searches the system relocatable library, if it is available. You must enter the CMS/DOS environment specifying the mode letter of the DOS system residence if you want to access the system libraries.

Linkage Editor Input

You can place the linkage editor control statements ACTION, PHASE, INCLUDE, and ENTRY in a CMS file with a filetype of DOSLNK. When you use the INCLUDE statement, you may specify the filename of a CMS TEXT file or the name of a module in a DOS relocatable library:

```
INCLUDE XYZ
```

or you may use the INCLUDE control statement to indicate that the object code follows:

```
INCLUDE
(CMS TEXT file)
```

A typical DOSLNK file, named CONTROL DOSLNK, might contain the following:

```
ACTION REL
PHASE PROGMAIN,S
INCLUDE SUBA
PHASE PROGA,*
INCLUDE SUBB
```

When you issue the command:

```
doslked control
```

the linkage editor searches the following for the object files SUBA and SUBB:

- A DOS private relocatable library, provided you have issued the ASSGN and DLBL commands to identify it:

  ```
  assgn sysrlb d
  dlbl ijsysrl d dsn ? (sysrlb
  ```

- Your CMS disks for files with filenames SUBA and SUBB and a filetype of TEXT
- The system relocatable library located on the DOS system residence volume (if it is available).

Link-editing TEXT Files

When you want to link-edit individual CMS TEXT files, you can insert linkage editor control statements in the file using the editor and then issue the DOSLKED command:

```
xedit rtnb text
input include rtnc
file
doslked rtnb mydoslib
```

When the above DOSLKED command is executed, the CMS file RTNB TEXT is used as linkage editor input, as long as there is no file named

RTNB DOSLNK. The ACTION statement, however, is not recognized in TEXT files.

You can also link-edit relocatable modules directly from a DOS system or private relocatable library, provided that you have identified the library. If you do this, however, you cannot directly provide control statements for the linkage editor.

To link-edit a relocatable module from a DOS private library and add linkage editor control statements to it, you could use this procedure:

1. Identify the library and use the RSERV command to copy the relocatable module into a CMS TEXT file. In this example, the module RTNC is to be copied from the library OBJ.MODS:

   ```
   assgn sysrlb e
   dlbl ijsysrl e dsn obj mods (sysrlb
   rserv rtnc
   ```

2. Create a DOSLNK file, insert the linkage editor control statements, and copy the TEXT file created in step 1 into it using the GET subcommand:

   ```
   xedit rtnc doslnk
   input action rel
   get rtnc text a
   file
   ```

3. Invoke the linkage editor with the DOSLKED command:

   ```
   doslked rtnc mydoslib
   ```

Alternatively, you could create a DOSLNK file with the following records: DOSLNK file

```
ACTION REL
INCLUDE RTNC
```

and link-edit the module directly from the relocatable library. If you do not need a copy of the module on a CMS disk, you might want to use this method to conserve disk space.

When the linkage editor is reading modules, it may encounter a blank card at the end of a file or a * (comment) card at the beginning of a file. In either case, the linkage editor issues a warning message indicating an invalid card, but it continues to complete the link-edit.

Linkage Editor Output: CMS DOSLIBs

The CMS/DOS linkage editor always places the link-edited executable phase in a CMS library with a filetype of DOSLIB. You should specify the filename of the DOSLIB when you enter the DOSLKED command:

```
doslked prog0 templib
```

where PROG0 is the relocatable module you are link-editing and TEMPLIB is the filename of the DOSLIB.

If you do not specify the name of a DOSLIB, the output is placed in a DOSLIB that has the same name as the DOSLNK or TEXT file being link-edited. In the above example, a CMS DOSLIB is created named TEMPLIB DOSLIB, or, if the file TEMPLIB DOSLIB already exists, the phase PROG0 is added to it.

DOSLIBs can contain relocatable core image phases suitable for execution in CMS/DOS. Before you can access phases in them, you must identify them to CMS with the GLOBAL command:

```
global doslib templib permlib
```

When CMS is searching for executable phases, it searches all DOSLIBs specified on the last GLOBAL DOSLIB command. If you have named a number of DOSLIBs or if any particular DOSLIB is very large, the time required for CMS to fetch and execute the phase increases. You should use separate DOSLIBs for executable phases, whenever possible. Then specify only the DOSLIBs you need on the GLOBAL command.

When you link-edit a module into a DOSLIB that already contains a phase with the same name, the directory entry is updated to point to the new phase. However, the space that was occupied by the old phase is not reclaimed. You should periodically issue the command:

```
doslib comp templib
```

to compress the DOSLIB and delete unused space. TEMPLIB is the filename of the DOSLIB.

Linkage Editor Maps

The DOSLKED command also produces a linkage editor map. It writes into a CMS file with a filename specified on the DOSLKED command line and a filetype of MAP. The filemode is always A5. If you do not want a linkage editor map, use the NOMAP option on the ACTION statement in a DOSLNK file.

Executing Programs in CMS/DOS

After you have assembled or compiled a source program and link-edited the TEXT files, you can execute the phases in your CMS virtual machine. You may not, however, be able to execute all your DOS programs directly in CMS. There are a number of execution-time restrictions placed on CMS/DOS programs. You cannot execute a program that uses:

- Multitasking
- More than one partition
- Teleprocessing
- ISAM macros to read or write files

- CMS module files created by DOS programs
- EC mode PSWs.

The above is only a partial list representing those restrictions with which you might be concerned. For a complete list of restrictions, see the *VM/SP Planning Guide and Reference.* See also the usage notes of the FETCH command in the *VM/SP CMS Command Reference.*

Executing DOS Phases

You can load executable phases into your CMS virtual machine using the FETCH command. Phases must be link-edited with ACTION REL before you load them. When you issue the FETCH command, you specify the name of the phase to be loaded:

```
fetch myprog
```

Then you can begin executing the program by issuing the START command:

```
start
```

Or, you can fetch a phase and begin executing it with a single command:

```
fetch prog2 (start
```

When you use the FETCH command without the START option, CMS issues a message telling you at what virtual storage address the phase is loaded:

```
PHASE PROG2 ENTRY POINT AT LOCATION 020000
```

Location X'20000' is the starting address of the user program area for CMS. Relocatable phases are always loaded starting at this address unless you specify a different address using the ORIGIN option of the FETCH command:

```
fetch prog3 (origin 22000
start
```

The program PROG3 executes beginning at location 22000 in the CMS user program area.

Search Order for Executable Phases

When you execute the FETCH command, CMS searches for the phase name you specify in the following places:

1. In a DOS private core image library on a DOS disk. If you have a private library you want searched for phases, you must identify it using the ASSGN and DLBL commands using the logical unit SYSCLB:

   ```
   assgn sysclb d
   dlbl ijsyscl d dsn ? (sysclb
   ```

When you enter the DLBL command with the ? operand, you are prompted to enter the DOS file-id.

2. In CMS DOSLIBs on CMS disks. If you want DOSLIBs searched for phases, you must use the GLOBAL command to identify the DOSLIBs to CMS/DOS:

   ```
   global doslib templib mylib
   ```

   You can specify up to 63 DOSLIBs on the GLOBAL command line.

3. On the DOS system residence core image library. If you want the system core image library searched you must have entered the CMS/DOS environment specifying the mode letter of the system residence:

   ```
   set dos on z
   ```

When you want to fetch a core image phase that has copies in both the core image library and a DOSLIB, and you want to fetch the copy from the CMS DOSLIB, you can bypass the core image library by entering the command:

```
assgn sysclb ua
```

When you need to use the core image library, enter:

```
assgn sysclb c
```

where C is the mode letter of the system residence volume. You do not need to reissue the DLBL command to identify the library.

Making I/O Device Assignments

If you are executing a program that performs I/O, you can use the ASSGN command to relate a system or programmer logical unit to a real I/O device:

```
assgn syslst printer
assgn sys052 reader
```

In this example, your program is going to read input data from your virtual card reader. The output print file is directed to your virtual printer. If you want to reassign these units to different devices, you must be sure that the files have been defined as device independent.

If you assign a logical unit to a disk, you should identify the file by using the DLBL command. On the DLBL command, you must always relate the DLBL to the system or programmer logical unit previously specified in an ASSGN command:

```
assgn sys015 b
dlbl myfile b dsn ? (sys015
```

When you enter the DLBL command with the ? operand you are prompted to enter the DOS file-id.

You must issue all of the ASSGN and DLBL commands necessary for your program's I/O before you issue the FETCH command to load the program phase and to begin executing.

Specifying a Virtual Partition Size

For most of the programs that you execute in CMS, you do not need to specify how large a partition you want those programs to execute in. When you issue the START command or use the START option on the FETCH command, CMS calculates how much storage is available in your virtual machine and sets a partition size. CMS calculates how much storage is available in the following manner:

FREELOWE - (MAINHIGH + (4096 * FRERESPG))

where:

FREELOWE	equals the low extent of allocated storage obtained from the top of virtual storage downwards via the DMSFREE system request.
MAINHIGH	equals the high extent of allocated storage obtained from the low virtual storage upwards via the GETVIS user request for storage.
FRERESPG	equals the amount of storage to be reserved for subsequent system requests, in pages.

In some instances, you may want to control the partition size:

- For performance considerations
- Because the default may not leave enough free storage to satisfy the GETVIS requests issued by the DOS program or the access method services function being executed.

You can set the partition size with the DOSPART operand of the SET command. For example, after you enter the command:

```
set dospart 300k
```

all programs that you subsequently execute during this session execute in a 300K partition. In this way you can:

- Set a smaller partition size for programs that run better in smaller partitions.
- Set a smaller partition size to leave more free storage. If the reduction of the DOS partition does not free enough storage for the GETVIS requests, a larger virtual machine must be defined. If you enter:

```
set dospart off
```

CMS calculates a partition size when you execute a program. This is the default setting.

Note: The CMS partition, unlike the DOS partition, is used only for the loading and executing of programs invoked by the FETCH or LOAD commands. Areas allocated by GETVIS are assigned addresses outside the partition but within the user's virtual machine.

Setting the UPSI Byte

If your program uses the user program switch indicator (UPSI) byte, you can set it by using the UPSI operand of the CMS SET command. The UPSI byte is initially binary zeros. To set it to ones, enter:

```
set upsi 11111111
```

To reset it to zeros, enter:

```
set upsi off
```

Any value you set remains in effect for the duration of your terminal session unless you reload CMS (with the IPL command).

Debugging Programs in CMS/DOS

You can debug your DOS programs in CMS/DOS using the facilities of CP and CMS. By executing your programs interactively, you can determine the cause of an error or program abend, correct it, and attempt to execute a program again. The CP and CMS debugging facilities are described in *VM Diagnosis Guide.*

Using EXEC Procedures in CMS/DOS

During your program development and testing cycle, you may want to create EXEC procedures to contain sequences of CMS commands that you execute frequently. For example, if you need a number of MACLIBs, DOSLIBs, and DLBL definitions to execute a particular program, you might have an EXEC procedure as follows:

```
/* EXEC to set up environment to run program TESTA */

signal on error
global maclib testlib dosmac
assemble testa
print testa listing
doslked testa testlib
global doslib testlib proglib
access 200 e
assgn sys010 e
push dos.test3.stream.beta
dlbl inddl e dsn ? '('sys010
assgn sys011 punch
cp spool punch to '*'
assgn sys012 a
dlbl outfile a cms test data '('sys012
signal off error
fetch testa '('start
select
  when rc = 100 then do

.
.
.
  end
  when rc = 200 then do

.
.
.
  end
  otherwise
    exit rc
end

Error:
  say 'Error occurred on line' sigl':' sourceline(sigl)
  exit rc
```

The 'signal on error' control statement in the EXEC procedure ensures that if an error occurs during any part of the EXEC, the remainder of the EXEC does not execute, and the 'Error' displays the line number where the error occurred as well as the actual command which gave the error.

Note: For the DLBL command entered with the DSN ? operand, you must stack the response (using 'push') before issuing the DLBL command.

When your program is finished executing, the REXX special variable RC indicates the contents of general register 15 at the time the program exited (the 'Return Code'). You can use this value to perform additional steps in your EXEC procedure. Additional steps are indicated in the preceding example by ellipses.

Hardware Devices Supported

CMS/DOS routines can read real DOS disks containing VSE data files and VSE private and system libraries. This read support is limited to the following disks supported by VSE:

- 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

The following devices, which are supported by VSE, are not supported by CMS/DOS:

- Card Readers: 1442, 2560P, 2560S, 2596, 3504, 5425P, and 5425S
- Printers: 2560P, 2560S, 3203 Models 1 and 2, 3525, 5203, 5425P, and 5425S
- Disks: 2311.

Also, CMS uses the CP spooling facilities and does not support dedicated unit record devices. Each CMS virtual machine supports only one virtual console, one reader, one punch, one printer, four tapes, and 26 disks. Programs that are executed in CMS/DOS are limited to the number of devices supported by CMS.

VSE Supervisor and I/O Macros Supported by CMS/DOS

CMS/DOS supports the VSE Supervisor macros and the SAM and VSAM I/O macros to the extent necessary to execute the DOS/VS COBOL Compiler, the DOS PL/I Optimizing Compiler, and DOS/VS RPG II Compiler under CMS/DOS. CMS/DOS supports VSE Supervisor macros described in the publication *VSE Macro Reference.*

Since CMS is a single-user system executing in a virtual machine with virtual storage, VSE operations, such as multi-tasking, that cannot be simulated in CMS are ignored.

The following information deals with the type of support that CMS/DOS provides in the simulation of VSE Supervisor and Sequential Access Method I/O macros. For a discussion of VSAM macros, see the section "CMS Support for OS and VSE/VSAM Functions."

Supervisor Macros

CMS/DOS supports physical IOCS macros and control program function macros for VSE. Figure 26 lists the physical IOCS macros and describes their support. Figure 27 lists the control program function macros and their support. Refer to *VM/SP System Logic and Problem Determination Guide Volume 2 (CMS)* for details of the macros' operation.

Macro	Support
CCB (command control block)	The CCB is generated.
IORB (input/output request block)	Supported for DASD I/O.
EXCB (execute channel program)	The REAL operand is not supported. All other operands are supported.
WAIT	Supported. Issued whenever your program requires an I/O operation (started by an EXCP macro) to be completed before execution of program continues.
SECTVAL (sector value)	Supported for VSAM.
OPEN/OPENR	Supported. Activates a data file.
LBRET (label processing return)	Not supported.
FEOV (forced end of volume)	Not supported.
SEOV (system end of volume)	Not supported.
CLOSE/CLOSER	Supported. Deactivates a data file.

Figure 26. Physical IOCS Macros Supported by CMS/DOS

Function/Macro	SVC. No. Dec Hex	Support
EXCP	0 0	Used to read from CMS or DOS/OS formatted disks.
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.
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.
FORCE DEQUEUE	3 3	Not supported. See note 2 on page 258.

Figure 27 (Part 1 of 9). 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.
MVCOM	5 5	Provides the user with a means of altering positions 12 through 23 of the partition communications region (BGCOM).
CANCEL	6 6	Cancels a VSE session either by a VSE program request or by a request from any of the CMS routines handling CMS/DOS.
WAIT	7 7	Used to wait on a CCB, IORB, ECB, or TECB. (Note that CMS/DOS does not support ECB's or TECB's). 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.
CONTROL	8 8	Temporarily return control from a $$B-transient to the problem program.
LBRET	9 9	Return to the $$B-transient after an SVC 8 was issued to give control to the problem program.
SET TIMER	10 A	No operation. Successful return code of 0 is given in R15. See note 1.
TRANS. RETURN	11 B	Return from a $$B-transient to the calling problem program.
JOB CONTROL 'AND'	12 C	Resets flags to 0 in the linkage control byte in BGCOM (communication region). If R1 = 0, bit 5 of JCSW4 (COMREG byte 59) is turned off.
JC FLAGS	13 D	Not supported. See note 2.
EOJ	14 E	Normally terminates execution of a problem program.
SYSIO	15 F	Not supported. See note 2.
PC STXIT	16 10	Establish or terminate linkage to a user's program check routine.
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 a program check.
IT STXIT	18 12	No operation. Successful return code of 0 is given in R15. See note 1.
IT EXIT	19 13	Not supported. See note 2.

Figure 27 (Part 2 of 9). SVC Support Routines and Their Operation

Function/Macro	SVC. No. Dec Hex	Support
OC STIXIT	20 14	No operation. Successful return code of 0 is given in R15. 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 R15. See note 1.
LOAD HEADER	23 17	Not supported. See note 2.
SETIME	24 18	No operation. Successful return code of 0 is given in R15. 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 purpose register 1.
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 24	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 R15. See note 1.
FREE	36 24	No operation. Successful return code of 0 is given in R15. See note 1.
AB STXIT	37 25	Establish or terminate linkage to a user's abnormal termination routine. Supported for OPINION=DUMP or NODUMP.
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 are turned 'on' by DMSDOS.

Figure 27 (Part 3 of 9). SVC Support Routines and Their Operation

Function/Macro	SVC. No. Dec Hex	Support
DEQ	41 29	No operation. Successful return code of 0 is given in R15. See note 1.
ENQ	42 2A	No operation. Successful return code of 0 is given in R15. 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 that the LIOCS was not generated to perform. The error message "unsupported function in a LIOCS routine" is issued, and the session is then terminated.
RETURN HEADER	51 33	Not supported. See note 2.
TTIMER	52 34	No operation. Successful return code of 0 is given in R15. See note 1. RO 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 to obtain free storage for scratch use or for obtaining an area where a relocatable program may be loaded. The PAGE, POOL, and SVA GETVIS options are ignored.
FREEVIS	62 3E	Used to return the free storage obtained via an earlier GETVIS call.

Figure 27 (Part 4 of 9). SVC Support Routines and Their Operation

Function/Macro	SVC. No. Dec Hex	Support
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.
CDLOAD	65 41	Used to load a relocatable VSAM phase into storage, unless the program has already been loaded.
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 R15. See note 1.
PFREE	68 44	No operation. Successful return code of 0 is given in R15. 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 R15. See note 1.
GETCBUF/ FREECBUF	72 48	Not supported. See note 2.
SETAPP	73 49	Not supported. See note 2.
PAGE FIX	74 4A	Not supported. See note 2.
SECTVAL	75 4B	Used by I/O routines to obtain a sector number for a 3330, 3330-11, 3340, or 3350 device.
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.

Figure 27 (Part 5 of 9). SVC Support Routines and Their Operation

Function/Macro	SVC. No. Dec Hex	Support
RELPAGE	85 55	Provides support for the RELPAG macro. At entry register 1 points to a list of 8-byte storage description area. 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, R15 holds return code as follows: R15 = 0 all areas have been released R15 = 2 one or more negative area lengths were specified R15 = 4 one or more pages to be released were outside the user storage area R15 =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 R15. See note 1.
PAGEIN	87 57	No operation. Successful return code of 0 is given in R15. See note 1.
TPIN	88 58	Not supported. See note 2.
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.
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 parameter list. Output is placed in an area provided by caller.
GETVCE	99 63	Caller requests device information about specific DASD. Information is returned in an output area pointed to from the parameter list. Register 1 contains a pointer to the parameter list on entry.
	100 64	Reserved

Figure 27 (Part 6 of 9). SVC Support Routines and Their Operation

Function/Macro	SVC. No. Dec Hex	Support
MODVCE	101 65	No operation. Successful return code of 0 is given in R15. 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 R15. 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 27 (Part 7 of 9). SVC Support Routines and Their Operation

Function/Macro	SVC. No. 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.

Figure 27 (Part 8 of 9). SVC Support Routines and Their Operation

Function/Macro	SVC. No. Dec Hex	Support
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 macro (define the lock). SVC 63 requests are mapped into a dummy DTL and processed by SVC 110.
		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 is given, and the SVC is treated as a "cancel".*

Figure 27 (Part 9 of 9). SVC Support Routines and Their Operation

Declarative Macros (Sequential Access Method I/O Macros)

CMS/DOS supports the following declarative macros:

- DTFCD - Types X'02' and X'04'
- DTFCN - Types X'03'
- DTFDI - Types X'33'
- DTFMT - Types X'10', X'11', X'12', and X'14'
- DTFPR - Types X'08'
- DTFSD - Types X'20'

The CDMOD, DIMOD, MTMOD, and PRMOD macros generate the logical IOCS routines that correspond with the declarative macros. For files on disk, the logical IOCS routines used during program execution reside in the CMSBAM DCSS and are not generated within the program. The operands that CMS/DOS supports for the DTF are also supported for the xxMOD macro. In addition, CMS/DOS supports three internal macros that the COBOL and PL/I compilers require: DTFCP (types X'31' and X'32'), CPMOD, and DTFSL.

DTFCD Macro -- Defines the File for a Card Reader

CMS/DOS does not support the ASOCFLE, FUNC, TYPEFILE=CMBND, and OUBLKSZ operands of the DTFCD macro. CMS/DOS ignores the SSELECT operand and any mode other than MODE=E. Figure 28 describes the DTFCD macro operands and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

Operand	Status	Description
DEVADDR=SYSxxx		Symbolic unit for reader-punch used for this file.
IOAREA1=xxxxxxxx	*	Name of the first I/O area.
ASOCFLE=xxxxxxxx	*	Not supported.
BLKSIZE=nnn	*	Length of one I/O area, in bytes. If omitted, 80 is assumed. If CTLCHR=YES is specified, BLKSIZE defaults to 81.
CONTROL=YES		CNTRL macro used for this file. Omit CTLCHR for this file. Does not apply to 2501.
CRDERR=RETRY	*	Retry if punching error is detected. Applies to 2520 and 2540 only. However, this situation is never encountered under CMS/DOS because hardware errors are not passed to the LIOCS module.
CTLCHR=xxx		(YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit CONTROL for this file.
DEVICE=nnnn	*	(2501, 2520, 2540, 3505, or 3525). If omitted, 2540 is default.
EOFADDR=xxxxxxxx		Name of your end-of-file routine.
ERROPT=xxxxxx	*	IGNORE, SKIP, or name. Applies to 3505 and 3525 only.
FUNC=xxx	*	Not supported.
IOAREA2=xxxxxxxx	*	If two output areas are used, name of second area.
IOREG=(nn)		Register number if two I/O areas are used and GET or PUT does not specify a work area. Omit WORKA.
MODE=xx	*	Only MODE=E is supported.
MODNAME=xxxxxxxx		Name of the logic module that is used with the DTF table to process the file.
OUBLKSZ=nn	*	Not supported.
RDONLY=YES	*	Causes a read-only module to be generated.
RECFORM=xxxxxx		(FIXUNB, VARUNB, UNDEF). If omitted, FIXUNB is default.
RECSIZE=(nn)	*	Register number if RECFORM=UNDEF.
SEPASMB=YES		DTFCD is to be assembled separately.
SSELECT=n	*	Ignored.

Figure 28 (Part 1 of 2). CMS/DOS Support of DTFCD Macro

Operand	Status	Description
TYPEFLE=	*	Input or output.
WORKA=YES		I/O records are processed in work areas instead of the I/O areas.

Figure 28 (Part 2 of 2). CMS/DOS Support of DTFCD Macro

DTFCN Macro - Defines the File for a Console

CMS/DOS supports all of the operands of the DTFCN macro. Figure 29 describes the operands of the DTFCN macro and their support under CMS/DOS. The status column is blank because the CMS/DOS and VSE support of DTFCN are the same.

Operand	Status	Description
DEVADDR=SYSxxx		Symbolic unit for the console used for this file.
IOAREA1=xxxxxxxx		Name of I/O area.
BLKSIZE=nnn		Length in bytes of I/O area (for PUTR macro usage, length of output part of I/O area). If RECFORM=UNDEF, maximum is 256. If omitted, 80 is default.
INPSIZE=nnn		Length in bytes for input part of I/O area for PUTR macro usage.
MODNAME=xxxxxxxx		Logic module name for this DTF. If omitted, IOCS generates a standard name. The logic module is generated as part of the DTF.
RECFORM=xxxxxx		(FIXUNB or UNDEF). If omitted, FIXUNB is default.
RECSIZE=(nn)		Register number if RECFORM=UNDEF. General purpose registers 2 through 12, enclosed in parentheses.
TYPEFLE=xxxxxx		(INPUT, OUTPUT, or CMBND). Input processes both input and output. CMBND must be specified for PUTR macro usage. If omitted, INPUT is default.
WORKA=YES		GET or PUT specifies work area.

Figure 29. CMS/DOS Support of DTFCN macro

DTFDI MACRO - Defines the File for Device Independence for System Logical Units

CMS/DOS supports most operands of the DTFDI macro. Figure 30 describes the operands of the DTFDI macro and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

Operand	Status	Description
DEVADDR=SYSxxx		(SYSIPT, SYSLST, SYSPCH, or SYSRDR). System logical unit. CMS/DOS issues an error message if the logical unit specified on the DTF does not match the logical unit specified on the corresponding DLBL command.
IOAREA1=xxxxxxxx		Name of the first I/O area.

Figure 30 (Part 1 of 2). CMS/DOS Support of DTFDI Macro

Operand	Status	Description
CISIZE=n	*	This operand specifies the control interval size for a DOS formatted FB-512 device assigned to a nonsystem file logical unit. This operand is ignored for count-key-data devices and CMS formatted disks.
EOFADDR=xxxxxxxx		Name of your end-of-file routine.
FBA=YES		This operand is not required and is ignored if specified.
ERROPT=xxxxxxxx		(IGNORE, SKIP, or name of your error routine). Prevents termination on errors.
IOAREA2=xxxxxxxx		If two I/O areas are used, name of second area.
IOREG2=(nn)		Register number. If omitted and two I/O areas are used, register 2 is default. General purpose registers 2 through 12, enclosed in parentheses.
MODNAME=xxxxxxxx		DIMOD name for this DTF. If omitted, IOCS generates a standard name. This operand is ignored with DASD. The SAM OPEN routines within the CMSBAM DCSS always load an IBM supplied logic module and link it to the DTF.
RDONLY=YES		Generates a read-only module. Requires a module save area for each routine using the module.
RECSIZE=nnn		Number of characters in record. Default values: 121 (SYSLST), 81 (SYSPCH), 80 (other).
SEPASMB=YES		DTFDI to be assembled separately.
TRC=YES	*	Not supported.
WLRERR=xxxxxxxx		Name of your wrong-length record routine.

Figure 30 (Part 2 of 2). CMS/DOS Support of DTFDI Macro

DTFMT Macro -- Defines the File for a Magnetic Tape

CMS/DOS does not support the ASCII, BUFOFF, HDRINFO, LENCHK, and READ=BACK operands of the DTFMT macro. Tape I/O operations are limited to reading in the forward direction.

You may use the FILABL operand in the DTFMT macro to specify that you have a standard tape label file, a nonstandard tape label file, or an unlabeled tape. The type of tape label processing depends on the option selected. See "Tape Labels in CMS" in the *VM/SP CMS User's Guide* for a complete description of tape label processing in CMS/DOS.

Figure 31 describes the DTFMT macro operands and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

Operand	Status	Description
BLKSIZE=nnnnn		Length of one I/O area in bytes (maximum = 32,767.
DEVADDR=SYSxxx		Symbolic unit for tape drive used for this file.
EOFADDR=xxxxxxxx		Name of your end-of-file routine.
FILABL=xxxx		(NO, STD, or NSTD). If NSTD specified, include LABADDR.
IOAREA1=xxxxxxxx		Name of first I/O area.
ASCII=YES	*	Not supported.
BUFOFF=nn	*	Not supported.
CKPTREC=YES		Checkpoint records are interspersed with input data records. IOCS bypasses checkpoint records.
ERREXT=YES		Additional errors and ERET are desired.
ERROPT=xxxxxxxx		(IGNORE, SKIP, or name of error routine). Prevents job termination on error records.
HDRINFO=YES	*	Not supported.
IOAREA2=xxxxxxxx		If two I/O areas are used, the name of the second area.
IOREG=(nn)		Register number. Use only if GET or PUT does not specify a work area or if two I/O areas are used. Omit WORKA. General purpose registers 2 through 12, enclosed in parentheses.
LABADDR=xxxxxxxx		Name of your label routine if FILABL=NSTD or if FILABL=STD and user-standard labels are processed.
LENCHK=YES	*	Not supported.
MODNAME=xxxxxxxx		Name of MTMOD logic module for this DTF. If omitted, IOCS generates standard name.
NOTEPNT=xxxxxx		(YES or POINTS). YES if NOTE, POINTW, POINTR, or POINTS macro is used. POINTS if only POINTS macro is used.
RDONLY=YES		Generate read-only module. Requires a module save area for each routine using the module.
READ=xxxxxxx	*	CMS/DOS only supports READ=FORWARD.
RECFORM=xxxxxx		(FIXUNB, FIXBLK, VARUNB, VARBLK, SPNUNB, SPNBLK, or UNDEF). For work files use FIXUNB or UNDEF. If omitted, FIXUNB is assumed.
RECSIZE=nnnn		If RECFORM=FIXBLK, number of characters in the record. If RECFORM=UNDEF, register number. Not required for other records. General purpose registers 2 through 12, enclosed in parentheses.
REWIND=xxxxxx		(UNLOAD or NORWD). Unload on CLOSE or end-of-volume, or prevent rewinding. If omitted, rewind only.
SEPASMB=YES		DTFMT is to be assembled separately.

Figure 31 (Part 1 of 2). CMS/DOS Support of DTFMT Macro

Operand	Status	Description
TPMARK=NO		Prevent writing a tapemark ahead of data records if FILABL=NSTD or NO.
TYPEFLE=xxxxxx		(INPUT, OUTPUT, or WORK). If omitted, INPUT is default.
VARBLD=(nn)		Register number, if RECFORM=VARBLK and records are built in the output area. General purpose registers 2 through 12 are enclosed in parentheses.
WLRERR=xxxxxxxx		Name of wrong-length record routine.
WORKA=YES		GET or PUT specifies a work area. Omit IOREG.

Figure 31 (Part 2 of 2). CMS/DOS Support of DTFMT Macro

DTFPR Macro - Defines the File for a Printer

CMS/DOS does not support the ASOCFLE, ERROPT=IGNORE, and FUNC operands of the DTFPR macro. Figure 32 describes the operands of the DTFPR macro and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

Operand	Status	Description
DEVADDR=SYSxxx		Symbolic unit for the printer used for this file.
IOAREA1=xxxxxxxx		Name for the first output area.
ASOCFLE=xxxxxxxx	*	Not supported.
BLKSIZE=nnn	*	Length of one output area, in bytes. If omitted, 121 is default.
CONTROL=YES		CNTRL macro used for this file. Omit CTLCHR for this file.
CTLCHR=xxx		(YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit CONTROL for this file.
DEVICE=nnnn	*	(1403, 1443, 3203, or 3211). If omitted, 1403 is default.
ERROPT=xxxxxxxx	*	RETRY or the name of your error routine for 3211. Not allowed for other devices. IGNORE is not supported.
FUNC=xxxx	*	Not supported.
IOAREA2=xxxxxxxx		If two output areas are used, name of second area.
IOREG=(nn)		Register number; if two output areas used and GET or PUT does not specify a work area. Omit WORKA.
MODNAME=xxxxxxxx		Name of PRMOD logic module for this DTF. If omitted, IOCS generates standard name.
PRINTOV=YES		PRTOV macro used for this file.

Figure 32 (Part 1 of 2). CMS/DOS Support of DTFPR Macro

Operand	Status	Description
RDONLY=YES		Generate a read-only module. Requires a module save area for each routine using the module.
RECFORM=xxxxxx		(FIXUNB, VARUNB, or UNDEF). If omitted, FIXUNB is default.
RECSIZE=(nn)		Register number if RECFORM=UNDEF.
SEPASMB=YES		DTFPR is to be assembled separately.
STLIST=YES		Use 1403 selective tape listing feature.
TRC=YES	*	Not supported.
UCS=xxx		(ON) process data checks. (OFF) ignores data checks. Only for printers with the UCS feature or 3203 or 3211. If omitted, OFF is default.
WORKA=YES		PUT specifies work area. Omit IOREG.

Figure 32 (Part 2 of 2). CMS/DOS Support of DTFPR Macro

DTFSD Macro - Defines the File for a Sequential DASD

CMS/DOS does not support the FEOVD, HOLD, and LABADDR operands of the DTFSD macro. Figure 33 describes the operands of the DTFSD macro and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

Operand	Status	Description
BLKSIZE=nnnn		Length of one I/O area, in bytes.
CISIZE=n	*	This operand specifies the control interval size for a DOS formatted FB-512 device assigned to a nonsystem file logical unit. This operand is ignored for count-key-data devices and CMS formatted disks.
EOFADDR=xxxxxxxx		Name of your end-of-file routine.
IOAREA1=xxxxxxxx		Name of first I/O area.
CONTROL=YES		This operand is ignored. CONTROL=YES is always included.
DELETFL=NO	*	If DELETFL=NO is specified, the work file is not erased. Otherwise, when the work file is closed, CMS/DOS erases it.

Figure 33 (Part 1 of 3). CMS/DOS Support of DTFSD Macro

Operand	Status	Description
DEVADDR=SYSnnn	*	Symbolic unit. This operand is optional. If DEVADDR is not specified, all I/O requests are directed to the logical unit identified on the corresponding CMS/DOS DLBL command. If a valid logical unit is specified with the DEVADDR operand of the DTF and a different, but also valid, logical unit is specified on the DLBL command, the unit specified on the DLBL command overrides the unit specified in the DTF. However, CMS/DOS issues an error message if a valid logical unit is specified in the DTF and no logical unit is specified on the corresponding DLBL command.
DEVICE=nnnn	*	This operand is ignored. The actual device type is determined by OPEN.
ERREXT=YES		Additional error facilities and ERET are desired. This operand is ignored. ERREXT=YES is always included.
ERROPT=xxxxxxxx		(IGNORE, SKIP, or name of error routine.) Prevents job termination on error records. Do not use SKIP for output files.
FEOVD=YES	*	Not supported.
HOLD=YES	*	Not supported. HOLD=YES is specified for DTFSD update or work files to provide a track hold capability. However, the CMS/DOS open routine sets the track hold bit off and bypasses track hold processing.
IOAREA2=xxxxxxxx		If two I/O areas are used, name of second area.
IOREG=(nn)		Register number. Use only if GET or PUT does not specify work area or if two I/O areas are used. Omit WORKA.
LABADDR=xxxxxxxx	*	Not supported.
MODNAME=xxxxxxxx		This operand is not required. If specified, it is ignored. The SAM OPEN routines within the CMSBAM DCSS always load an IBM supplied logic module and link it to the DTF.
NOTEPNT=xxxxxxxx		Indicates that NOTE, POINTR, POINTW, and POINTS are used. This operand is ignored. NOTEPNT=YES is always included.
RDONLY=YES		This operand is not required and is ignored if specified. RDONLY=YES is always included.
PWRITE=YES	*	For a DOS formatted FB-512 disk, this operand specifies that for output operations a physical write occurs for every logical block. This operand is ignored for count-key-data devices and CMS formatted disks. DOS formatted FB-512 disks are not supported for output.

Figure 33 (Part 2 of 3). CMS/DOS Support of DTFSD Macro

Operand	Status	Description
RECFORM=xxxxxx		(FIXUNB, FIXBLK, VARUNB, SPNUNB, SPNBLK, VARBLK, or UNDEF). If omitted, FIXUNB is assumed. For work files, use FIXUNB or UNDEF. Although work files contain fixed-length unblocked records, the CMS file system handles work UNDEF files as variable-length record files. If you specify FIXBLK, VARBLK, or UNDEF when creating a CMS file on a CMS disk, CMS writes the file in variable-length format. The LISTFILE command would show the file as V format. If you specify FIXUNB when creating a CMS file on a CMS disk, CMS writes the file in fixed-length format.
RECSIZE=nnnnn		If RECFORM=FIXBLK, number of characters in record. If RECFORM=SPNUNB, SPNBLK, or UNDEF, register number. Not required for other records.
SEPASMB=YES		DTFSD is to be assembled separately.
TRUNCS=YES		RECFORM=FIXBLK or TRUNC macro used for this file.
TYPEFLE=xxxxxx		(INPUT, OUTPUT, or WORK). If omitted, INPUT is assumed.
UPDATE=YES		Input file or work file is to be updated.
VARBLD=(nn)		Register number if RECFORM=VARBLK and records are built in the output area. Omit if WORKA=YES.
VERIFY=YES		Check disk records after they are written.
WLRERR=xxxxxxxx		Name of your wrong-length record routine.
WORKA=YES		GET or PUT specifies work area. Omit IOREG. Required for RECFORM=SPNUNB or SPNBLK.

Figure 33 (Part 3 of 3). CMS/DOS Support of DTFSD Macro

Imperative Macros (Sequential Access Method I/O Macros)

CMS/DOS supports the following imperative macros:

- *Initialization macros*: OPEN and OPENR
- *Processing macros*: GET, PUT, PUTR, RELSE, TRUNC, CNTRL, ERET, and PRTOV.

  Note: No code is generated for the CHNG macro.
- *Work file macros for tape and disk*: READ, WRITE, CHECK, NOTE, POINTR, POINTW, and POINTS.
- *Completion macros*: CLOSE and CLOSER.

CMS/DOS supports workfiles containing fixed-length unblocked records and undefined records. Disk work files are supported as single volume, single pack files. Normal extents and split extents are both supported.

VSE Transient Routines

CMS/DOS simulates the VSE transients that are fetched by macro expansion or by the LIOCS modules. These simulation routines contain enough of the transient's function to support the DOS/VSE COBOL compiler and DOS PL/I Optimizing compiler.

The following VSE transients are simulated by CMS/DOS.

$$BOPEN	Fetched by the VSE OPEN macro expansion or by the VSE LIOCS modules. $$BOPEN performs DTF initialization, dependent upon the device type, to ready the file for I/O operations. At entry to $$BOPEN, register 0 points to a list of fullword addresses containing a pointer to the DTFs. $$BOPEN checks for supported DTF types, and initializes DTFs in accordance with the device type. In the case of tape data files, default DLBLs with the NOCHANGE option are issued. (The CMS STATE command is issued to verify the existence of the input files on disk.) If a VSAM file is being opened (Byte 20 = X'28' in the ACB), control is passed to the VSAM OPEN routine. When opening DTFSD files for output or DTFCP/DTFDI disk files for output, if a file exists on a CMS disk with the same filename, filetype, and filemode, the file is erased. If a SAM disk file is being opened, DTF initialization is performed by involving the simulated VSE OPEN routines that reside in the CMSBAM DCSS.
$$BOPNLB	Fetched by COBOL Compiler Phase 00 to read the appropriate system or private source statement library directory record and to determine whether or not active members are present for the library.
$$BCLOSE	Fetched by VSE CLOSE macro expansion to deactivate a file.
$$BDUMP	Fetched when an abnormal termination condition is encountered. Control is not passed to a STXIT routine. CMS/DOS performs a CP dump to a virtual printer. The routine is canceled.
$$BOPENR	Fetched by a VSE OPENR macro expansion. The function of $$BOPENR is to relocate all DTF table address constants from the assembled addresses to executable storage addresses. At entry to $$BOPENR, register 0 points to an assembled address constant followed by a list of DTF addresses tables that require address modification.
$$BOPNR3	Fetched by $$BOPENR to relocate all DTF table address constants for unit record DTFs.

$$BOPNR2	Fetched by $$BOPNR3 to relocate all DTF table address constants for DTFDI or DTFCP.
$$BOSVLT	Fetched via SVC 2 by the simulated VSE OPEN/CLOSE routines in the CMSBAM DCSS. $$BOSVLT performs clean-up and transition functions when processing by the simulated VSE routines in the CMSBAM DCSS is complete.

EXCP Support in CMS/DOS

CMS/DOS simulates the EXCP (execute channel program) routines to the extent necessary to support the LIOCS routines described in the preceding section, "VSE Supervisor and I/O Macros Supported by CMS/DOS."

Because CMS/DOS uses the VSE LIOCS routines, it must simulate all I/O at the EXCP level. The EXCP simulation routines convert all the I/O in the CCW format to CMS physical I/O requests. That is, CMS macros (such as RDBUF/WRBUF, CARDRD/CARDPH, PRINTIO, and WAITRD/TYPLIN) replace the CCW strings. If CMS/DOS is reading from DOS disks, I/O requests are handled via the DIAGNOSE interface.

When an I/O operation completes, CMS/DOS posts the CCB or IORB with the CMS return code. Partial RPS (rotational position sensing) support is available for I/O operations to CMS disks because CMS uses RPS in its channel programs. However, RPS is not supported when real DOS disks are read.

VSE Supervisor Control Blocks Simulated by CMS/DOS

CMS/DOS supports VSE program development and execution for a single partition: the background partition. Because CMS/DOS does not support foreground partitions, it also does not simulate the associated control blocks and fields for foreground partitions. CMS/DOS does simulate the following VSE supervisor control blocks:

- ABTAB -- Abnormal Termination Option Table
- BBOX -- Boundary Box
- BGCOM -- Background Partition Communication Region
- EXCPW -- Work area for module DMSXCP
- FICL -- First in Class
- LUB -- Logical Unit Block
- NICL -- Next in Class
- PCTAB -- Program Check Option Table
- PIBTAB -- Program Information Table
- PIB2TAB -- Program Information Block Table Extension
- PUB -- Physical Unit Block
- PUBOWNER -- Physical Unit Block Ownership Table
- SYSCOM -- System Communication Region
- TCB -- Task Control Block

- LOCTAB -- LOCK/UNLOCK Resource Table
- DIB -- Disk Information Block.

For detailed descriptions of CMS/DOS control blocks, refer to the *VM/SP Data Areas and Control Block Logic Volume 2 (CMS).*

CMS/DOS User Considerations and Responsibilities

A critical design assumption of CMS/DOS is that installations that use CMS/DOS for VSE program development also use and have available a VSE system.

You should consider several factors if you plan to use CMS/DOS. The following sections describe some of the user considerations and responsibilities.

VSE System Generation and Updating Considerations

The CMS/DOS support in CMS may use a real VSE system pack. CMS/DOS provides the necessary path and then fetches VSE logical transients and system routines directly as well as the DOS/VS COBOL and DOS PL/I Optimizing compilers from the VSE system or private core image libraries.

It is your responsibility to order a VSE system and then generate it. Also, if you plan to use DOS compilers, you must order the current level of the DOS/VS COBOL compiler and DOS PL/I Optimizing compiler and you must install them on the same VSE system.

When you install the compilers on the VSE system, you must link-edit all the compiler modules as relocatable phases using the following linkage editor control statement:

```
ACTION REL
```

You can place the link-edited phases in either the system or the private core image library.

When you later invoke the compilers from CMS/DOS, the library (system or private) containing the compiler phases must be identified to CMS. You identify all the system libraries to CMS by coding the filemode letter that corresponds to that VSE system disk on the SET DOS ON command when you invoke the CMS/DOS environment. You identify a private library by coding ASSGN and DLBL commands that describe it. The VSE system and private disks must be linked to your virtual machine and accessed before you issue the commands to identify them for CMS.

CMS/DOS has no effect on the update procedures for VSE, COBOL, or DOS PL/I. Normal update procedures for applying IBM-distributed coding changes apply.

For detailed information on how to generate VM/SP with CMS/DOS, refer to the publication *VM/SP Planning Guide and Reference* and the *VM/SP Installation Guide.*

VM/SP Directory Entries

The VSE system and private libraries are accessed in read-only mode under CMS/DOS. If more than one CMS virtual machine is using the CMS/DOS environments you should update the VM/SP directory entries so that the VSE system residence volume and the VSE private libraries are shared by all the CMS/DOS users.

The VM/SP directory entry for one of the CMS virtual machines should contain the MDISK statements defining the VSE volumes. The VM/SP directory entries for the other CMS/DOS users should contain LINK statements.

For example, assume the VSE system libraries are on cylinders 0 through 149 of a 3330 volume labeled DOSRES. And, assume the VSE private libraries are on cylinders 0 through 99 of a 2314 volume labeled DOSPRI. Then, one CMS machine (for example, DOSUSER1) would have the MDISK statements in its directory entry.

```
USER DOSUSER1 password 320K 2M G
      .
      .
      .
MDISK 331 3330 0 150 DOSRES R rpass
MDISK 231 2314 0 100 DOSPRI R rpass
```

All the other CMS/DOS users would have links to these disks. For example

```
LINK DOSUSER1 331 331 R
LINK DOSUSER1 231 231 R
```

When the VSE System Must be Online

Most of what you do in the CMS/DOS environment for VSE program development requires that the VSE system pack and/or the VSE private libraries be available to CMS/DOS. In general, you need these VSE volumes whenever:

- You use the DOS/VS COBOL compiler or DOS/PLI Optimizing compiler. The compilers are executed from the system or private core image libraries.
- Your source programs contain COPY, LIBRARY, %INCLUDE, or CBL statements. These statements copy books from your system or from the private source statement library.
- You invoke one of the library programs: DSERV, RSERV, SSERV, PSERV, or ESERV.

- You execute VSE programs that use LIOCS modules. CMS/DOS fetches most of the LIOCS routines for non-disk files directly from VSE system or private libraries.

A VSE system pack is usable when it is:

- Defined for your virtual machine
- Accessed
- Specified, by mode letter, on the SET DOS ON command.

A VSE private library is usable when it is:

- Defined for your virtual machine
- Accessed
- Identified via ASSGN and DLBL commands.

Performance

Although you can use the CMS/DOS library services to place the DOS/VS COBOL compiler, DOS PL/I compiler, and ESERV program in a CMS DOSLIB, it is recommended that you do not use this method with 800-byte format CMS disks. CMS/DOS can fetch these directly from the VSE system or private libraries faster than from a DOSLIB on 800-byte format CMS disks. Fetch time from DOSLIBs on 512, 1K-, 2K-, or 4K-byte format CMS disks is approximately equivalent to that of VSE system or private libraries.

Execution Considerations and Restrictions

The CMS/DOS environment does not support the execution of VSE programs that use:

- Teleprocessing or indexed sequential (ISAM) access methods. CMS/DOS supports only the sequential (SAM) and virtual storage (VSAM) access methods.

- Multi-tasking. CMS/DOS supports only a single partition, the background partition.

CMS/DOS can be executed in a CMS Batch Facility virtual machine. If any of the VSE programs that are executed in the batch machine read data from the card reader, you must ensure that the end-of-data indication is recognized. Be sure that (1) the program checks for end of data and (2) a /* record follows the last data record.

If there is an error in the way you handle end of data, the VSE program could read the entire batch input stream as its own data. The result is that jobs sent to the batch machine are never executed and the VSE program reads records that are not part of its input file.

Chapter 10. Using Access Method Services and VSAM under CMS and CMS/DOS

This section describes how you can use CMS to create and manipulate VSAM catalogs, data spaces, and files on OS and DOS disks using access method services. The CMS support is based on VSE and VSE/VSAM. This means that if you are an OS VSAM user and plan to use CMS to manipulate VSAM files, you are allowed to use those functions of access method services that are available under the access method services portion of VSE/VSAM. The control statements you can use are described in the publication *Using VSE/VSAM Commands and Macros.*

You can use CMS to:

- Execute the access method services utility programs for VSAM and SAM data sets on OS and DOS disks and minidisks. CMS can both read and write VSAM files using access method services.
- Compile and execute programs that read and write VSAM files from VSE programs.
- Compile and execute programs that read and write VSAM files from OS programs.

VSAM files written by CMS are written using VSE/VSAM. Certain files written under CMS cannot be used directly by OS/VS VSAM. For information relative to compatibility between VSE/VSAM and OS/VS VSAM files, you should refer to the *VSE/VSAM General Information Manual.* The CMS commands normally used to manipulate CMS files are not applicable to VSAM files, however. This includes such commands as PRINT, TYPE, EDIT, COPYFILE, and so on.

Under CMS, VSAM data sets can span up to 10 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 by CMS are not in the CMS file format. Therefore, CMS commands currently used to manipulate CMS files cannot be used for VSAM data sets read or written by CMS. A VSAM data set created by CMS (using VSE/VSAM) has a file format compatible with OS VSAM data sets as long as the physical record size of the data set is .5K, 1K, 2K, or 4K. For complete information on OS/VS VSAM and VSE/VSAM data set compatibility, see the *VSE/VSAM General Information Manual.*

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 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 with the Device Support Facility; the CMS FORMAT command can not be used.

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

This section provides information on using the CMS AMSERV command. The CMS AMSERV command allows you to execute access method services. Information is provided on using VSAM macros in CMS. The discussion is divided as follows:

- "Using the AMSERV command" contains general information.
- "Manipulating OS and DOS Disks for Use With AMSERV" describes how to use CMS commands with OS and DOS disks.
- "Defining DOS Input and Output Files" is for CMS/DOS users only.
- "Defining OS Input and Output Files" is for OS users only.
- "Using AMSERV Under CMS" includes notes and examples showing how to perform various access method services functions in CMS.
- "VSE/VSAM Macros" describes the macros and their support in CMS.
- "OS/VSAM Macros" describes the OSVSAM MACLIB supplied with CMS.
- "Hardware Devices Supported" describes the disks supported by VSE that VSAM data sets in CMS can use.

Executing VSAM Programs Under CMS

The commands that are used to define input and output data sets for Access Method Services (DLBL) and for CMS/DOS users (ASSGN) are also used to identify VSAM input and output files for program execution. Information on executing programs under CMS that manipulate VSAM files is contained in the licensed program documentation for the language processors. These publications are listed in the *VM/SP Introduction.*

Restrictions on the use of access method services and VSAM under CMS for OS and DOS users are listed in the *VM/SP CMS Command Reference.* The *VM/SP CMS Command Reference* also contains complete CMS and CMS/DOS command formats, operand descriptions, and responses for each of the commands described here.

When you are going to execute VSAM programs in CMS or CMS/DOS, you should remember to issue the DLBL commands to identify the master catalog and any other program input or output files you need to define.

Since VSE/VSAM Release 2, VSE/VSAM has reduced its dependency on explicit ASSGN, EXTENT, and DLBL information. In many cases, you no longer need to specify this information. Identification of the master catalog within CMS, however, still requires ASSGN and DLBL commands. For complete information concerning the ASSGN, DLBL, and EXTENT requirements, refer to the *VSE/VSAM Programmer's Reference.*

Note: For ASSGN, EXTENT, and DLBL requirements for multivolume files, refer to "Defining DOS Input and Output Files" on page 284 and"Identifying Existing Multivolume Files" on page 292.

In the discussion that follows, ASSGN, DLBL, and EXTENT information is included even though it may not be required.

Opening an ACB with a MACRF=ADR and subsequently issuing a GET or a PUT with KEYED ACCESS specified in the RPL when SHAREOPTION (4) is specified is not allowed in VSE/VSAM Release 2. Likewise, opening an ACB with KEYED ACCESS and subsequently issuing a GET or a PUT with MACRF=ADR specified in the RPL when SHAREOPTION (4) is specified is not allowed. Please refer to *Using VSE/VSAM Commands and Macros* for more information.

VSE/VSAM supports the functions that were previously supported as well as the following enhancements:

- Volume ownership is enhanced so that multiple catalogs may own space in the same DASD volume if only one recoverable catalog owns space on the volume and only if one catalog resides on the volume.

- You can verify the syntax of the AMS commands without actually executing them by using the SYNCHK parameter of the AMS PARM command.

- Using the IGNOREERROR parameter of the AMS DELETE command, you can delete incomplete catalog information that may have resulted from a system failure during DEFINE or DELETE processing. When you specify the IGNOREERROR parameter of the AMS DELETE command, the PRINT option must be used on the CMS AMSERV command to send the listing to the virtual printer.

- By issuing the CMS CATCHECK command, a CMS VSAM user (with or without DOS set ON) may invoke the VSE/VSAM Catalog Check Service Aid to verify a complete catalog structure.

The AMSERV Command

In CMS, you execute access method services utility programs with the AMSERV command. The basic format is:

```
amserv filename
```

`filename` is the name of a CMS file containing the control statements for access method services.

Note: Throughout the remainder of this section the term "AMSERV" is used to refer to both the CMS AMSERV command and the OS/VS or VSE/VSAM access method services, except where a distinction is being made between CMS and access method services.

You create an AMSERV file with the CMS editor using a filetype of AMSERV and any filename you want. For example:

```
xedit mastcat amserv
```

The editor recognizes the filetype of AMSERV and so automatically sets the zone for your input lines at columns 2 and 72. The sample AMSERV file being created in the example above, MASTCAT AMSERV, might contain the following control statements:

```
 DEFINE MASTERCATALOG (NAME (MYCAT) -
    VOLUME (123456) CYL(2) -
    FILE (IJSYSCT) )
```

Note: The syntax of the control statements must conform to the rules for access method services, including continuation characters and parentheses. The only difference is that the AMSERV file does not contain a "/*" for a termination indicator.

Before you can execute the DEFINE control statement in this AMSERV example, you must define the output file, using the ddname IJSYSCT. You can do this using the DLBL command, if required by VSE/VSAM. Since the exact form required in the DLBL command varies according to whether you are an OS or a DOS user, separate discussions of the DLBL command are provided later in this section. All of the following examples assume that any disk data set or file that you are referencing with an AMSERV command was defined by a DLBL command, if required by VSE/VSAM.

When you execute the AMSERV command, the AMSERV control statement file can be on any accessed CMS disk. You do not need to specify the filemode. If you are a DOS user, you do not need to assign SYSIPT. The task of locating the file and passing it to access method services is performed by CMS.

AMSERV Output Listings

When the AMSERV command is finished processing, you receive the CMS ready message. If there was an error, the return code (from register 15) is displayed following the "Ready". For example:

```
Ready(00008);
```

If you are receiving the long form of the ready message, it appears:

```
Ready(00008); T=0.01/0.11 10:50:23
```

If you receive a ready message with an error return code, you should examine the output listing from AMSERV to determine the cause of the error.

AMSERV output listings are written in CMS files with a filetype of LISTING. By default, the filename is the same as that of the input AMSERV file. For example, if you have executed:

```
amserv mastcat
```

and the CMS ready message indicates an error return code, you should examine the file MASTCAT LISTING. Edit the file MASTCAT LISTING and issue the following LOCATE subcommand twice:

```
locate /idc
```

to find the character string IDC will position you in the LISTING file at the first access method services message.

The publication *VSE/VSAM Messages and Codes* lists and explains all of the messages generated by access method services together with the associated return and reason codes.

If you need to make changes to control statements before executing the AMSERV command again, use the CMS editor to modify the AMSERV input file.

If you execute the same AMSERV file a number of times, each execution results in a new LISTING file that replaces any previous listing file with the same filename.

Controlling AMSERV Command Listings

When you use AMSERV to print a VSAM file or to list catalog or recovery area contents using the PRINT, LISTCAT, or LISTCRA control statements, the output is written in a listing file on a CMS read/write disk.

If you only want a printed copy of the output listing, issue the AMSERV command with the PRINT option. For example,

```
amserv myfile (print
```

You might want to use this option if you are executing a PRINT or LISTCAT control statement and expect a very large output listing that you know cannot be contained on any of your disks.

If you want to save the output listing on disk and also print a copy, issue the AMSERV command without the PRINT option, and then use the CMS PRINT command to print the LISTING file.

If you issue the AMSERV command with no options, you get a CMS file with a filetype of LISTING and a filename equal to that of the AMSERV input file. This LISTING file is usually written on your A-disk, but if your A-disk is full or not accessed, it is written on any other read/write CMS disk you have accessed.

If there is not enough room on your A-disk or any other disk, the AMSERV command issues an error message saying that it cannot write the LISTING file. If this happens, the LISTING file created may be incomplete and you may not be able to tell whether or not access method services actually completed successfully. In this case, after you have cleared some space on a read/write disk, you may have to execute an AMSERV PRINT or LISTCAT function to verify the completion of the prior job.

LISTING files take up considerable disk space, so you should erase them as soon as you no longer need them.

Controlling the Filename of the Output Listing

You can also control the filename of the output listing file by specifying a second name on the AMSERV command line. For example,

```
amserv myfile myfile1
```

In this example, the input file is MYFILE AMSERV and the output listing is placed in a file named MYFILE1 LISTING. A subsequent execution of this same AMSERV file:

```
amserv myfile myfile2
```

creates a second listing file, MYFILE2 LISTING, so that the listing created from the first execution is not erased.

Manipulating OS and DOS Disks for Use with AMSERV

To use CMS VSAM and AMSERV, you can have OS or DOS disks in your virtual machine configuration. They can be assigned in your directory entry, or you can link to them using the CP LINK command. You must have read/write access to them in order to execute any AMSERV function or VSAM program that requires opening the file for output or update.

Before you can use an OS or DOS disk, you must access it with the CMS ACCESS command:

```
access 200 d
```

The response from the ACCESS command indicates that the disk is in OS or DOS format:

```
D (200) R/W - OS
   -- or --
D (200) R/W - DOS
```

You can write on these disks only through AMSERV or through the execution of a program writing VSAM data sets. Once an OS disk is used with AMSERV or VSAM, CMS considers it a DOS disk. Therefore, regardless of whether you are an OS user, when you access or request information about a VSAM disk, CMS indicates that it is a DOS disk. You can still use the disk in an OS or DOS system for VSAM data set processing. Although the format is not changed, the disk is still subject to any incompatibilities that can currently exist between OS and DOS disks.

Data and Master Catalog Sharing

There are two meanings of "sharing" that must be defined clearly with respect to the CMS support of VSAM. The first is that of the SHAREOPTION parameter found in the DEFINE (and ALTER) command for access method services.

The SHAREOPTION keyword enables the VSAM user to define how a component is shared within or across VSE partitions and VSE systems. Since CMS supports only a single partition environment, cross partition sharing has no meaning in the CMS environment. In addition, since CMS does not provide DASD sharing support, cross system sharing is not supported. Consequently, the SHAREOPTION parameter only has meaning within a CMS virtual machine (functional equivalent of a VSE partition).

The area of sharing most familiar to CMS users is that of disk (minidisk) read-sharing provided by CP. For the VSAM user under CMS, it is still possible to share disks in read-only mode in order to read-share VSAM components. However, there is a restriction with respect to the VSAM master catalog. That is, only one virtual machine may have the disk containing the master catalog in write status. This is necessary even if only read functions are being performed during the session. This is due to the master catalog updating read statistics at close time and, when necessary, writing a new control record in the catalog at open time.

Under CMS, it is possible to have the master catalog disk read-only. A programming modification (a bit in the ACB) was made to the DOS/VS VSAM code so that VSAM knows it is running under CMS. If this bit is on, VSAM will not write to the master catalog for either of the two cases described above. This allows one or more CMS virtual machines to share the VSAM master catalog. This assumes either no other virtual machine has the master catalog disk in write status or only one virtual machine (DOS, OS, or CMS) has it.

Multiple CMS users may have the VSAM master catalog disk in read-only status but only one virtual machine may have the same in write status. With respect to data set sharing, there is only read-sharing for the CMS user.

Disk Compatibility

Since the CMS VSAM support writes VSAM data sets to DOS disks, the question of disk compatibility is not one between CMS and DOS nor between CMS or OS but rather between DOS and OS disks. Because CMS actually uses VSE/VSAM for processing VSAM data sets, all disks used by CMS VSAM are DOS disks. For this reason, we need only discuss how DOS and OS disks are compatible.

In the format-4 DSCB, there is a bit in the VTOC indicators (byte 59, bit 0) defined by OS/VS to indicate (when OFF) that a format-5 label is included in the VTOC. This bit is always ON under VSE because DOS does not maintain the format-5 label. This technique allows OS/VS to realize when the format-5 is invalid and that it must recompute free space and rewrite the format-5 label.

Thus, if a disk originally was used under OS/VS, further allocation could occur under VSE but with the format-5 ignored. If the disk was then used under OS/VS and additional allocation performed, OS/VS would recognize the fact that the format-5 was not valid and would rewrite the format-5.

In terms of space allocation, DOS and OS disks are portable between the two systems. However, OS/VS must perform extra processing prior to using the disk if it intends to reallocate using the format-5.

DOS and OS disks containing VSAM data sets are no exception to this. OS and DOS disks containing VSAM data sets that are used under CMS are portable among all three systems. Since CMS uses the actual VSE/VSAM routines, all disks used under CMS to process VSAM data sets become DOS disks.

VSE/VSAM uses physical record sizes ranging from .5K bytes to 8K bytes. All multiples of .5K bytes between those two values are supported. OS/VS VSAM, however, only supports physical record sizes of .5K, 1K, 2K, and 4K. Therefore, some VSAM files written under CMS cannot be used directly by OS/VS VSAM.

Allocating Space

It is necessary to distinguish between two types of allocation under VSAM.

1. The actual space allocation on the disk
2. Allocation within the data set itself

Space for VSAM components must be allocated on the DASD using the DEFINE commands. You can only allocate space for the master catalog, a user catalog, a data space, and a UNIQUE cluster.

In defining the actual DASD space for components, there are parameters for the DEFINE SPACE command that allows the user to include a "secondary allocation" specification. These parameters are CYLINDERS, RECORDS, BLOCKS, and TRACKS. They have this secondary facility only as a syntactic compatibility with the OS/VS access method services commands. That is, VSE (and, therefore, CMS) does not perform secondary space allocation on a DASD.

The facility does exist under VSE (and CMS) to extend data or index components through already allocated data space, catalog extents, or UNIQUE cluster extents. Thus, the CYLINDERS, TRACKS, RECORDS, and BLOCKS parameters of the DEFINE commands for alternate indexes, clusters, and catalogs do not dynamically allocate DASD space but only extend a component through existing space.

Using VM/SP Minidisks

If you have a VM/SP minidisk in your virtual machine configuration, you can use it to contain VSAM files. Before you can use it, it must be formatted with the Device Support Facility program. When you request that a disk be added to your userid for use with VSAM files under CMS, you should indicate that it be formatted for use with OS or DOS. Or you can format it yourself using the Device Support Facility. How to do this is described under "Using Temporary Disks."

Note: If you are an OS user, you should be careful about allocating space for VSAM on minidisks. Once you have used CMS AMSERV to allocate VSAM data space on a minidisk, you should not attempt to allocate filetional space on that minidisk using an OS/VS system. OS does not recognize minidisks, and would attempt to format the entire disk pack and thus erase any data on it. To allocate additional space for VSAM, you should use CMS again.

Minidisk space allocation is fully described in the *VM/SP Planning Guide and Reference.*

The LISTDS Command

For OS or DOS disks or minidisks, you can use the LISTDS command to determine the extents of free space available for use by VSAM. You can also determine what space is already in use. You can use this information to supply the extent information when you define VSAM files.

The options used with VSAM disks are:

- EXTENT -- to find out what extents are in use
- FREE -- to find out what extents are available.

For example, if you have an OS disk accessed as a G-disk, and you enter:

```
listds g (extent
```

The response might look like:

```
EXTENT INFORMATION FOR 'VTOC' ON 'G' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
000 VTOC  0099 00  1881     0099 18  1899        19

EXTENT INFORMATION FOR 'PRIVAT.CORE.IMAGE.LIB' ON 'G' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
000 DATA  0000 01       1   0049 18   949       949

EXTENT INFORMATION FOR 'SYSTEM.WORK.FILE.NO.6' ON 'G' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
000 DATA  0050 00     950   0051 18   987        38
```

You could also determine the extent for a particular data set:

```
listds ? * (extent
```

CMS responds:

```
DMSLDS220R Enter dataset name:
```

Then, you can enter the file-id:

```
system.recorder.file
```

The response might look like:

```
EXTENT INFORMATION FOR 'SYSTEM RECORDER FILE' ON 'F' DISK:
SEQ TYPE  CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
000 DATA  0102 00  1938    0102 18   1956        19
002 DATA  0010 06   206    0010 08    208         3
```

LISTDS searches all minidisks accessed until it locates the specified data set. In this example, the data set occupies two separate extents on disk F. If the data set is a multivolume data set, extents on all accessed volumes are located and displayed.

If you want to find the free extents on a particular disk, enter:

```
listds g (free
```

The response might look like:

```
FREESPACE EXTENTS FOR 'G' DISK:
CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
0052 00    988    0052 01    989        2
0054 02   1028    0080 00   1520      493
0081 01   1540    0098 18   1880      341
```

You can use this information when you allocate space for VSAM files. If you enter:

```
listds * (free
```

CMS lists all the free space available on all of your accessed disks.

Using Temporary Disks

When you need extra space on a temporary basis for use with CMS VSAM and AMSERV, you can use the CP DEFINE command to create a temporary minidisk and then use the Device Support Facilities program to format it. Refer to the *Device Support Facilities User's Guide and Reference.* Once formatted and accessed, it is available to your virtual machine for the duration of your terminal session or until you detach it using the CP DETACH command. Remember that anything placed on a temporary disk is lost, so that you should copy output that you want to keep onto permanent disks before you log off.

Formatting a Temporary Disk

The example below shows a control statement file and an EXEC procedure that, together, can be used to format a minidisk using the Device Support Facility. For a complete description of the control statements used, refer to the *Device Support Facilities User's Guide and Reference.*

The input control statements for the Device Support Facility should be placed in a CMS file so that they can be punched to your virtual card reader. For this example, suppose the statements are in a CMS file named TEMP DSF:

```
INIT UNIT(198) DEVTYP(3340) PRG NVFY VOLID(123456) DVTOC(9,7,5) -
  MIMIC (MINI(10))
```

Note: The example above begins in column 2.

Now consider the CMS file named TEMPDISK EXEC:

```
/* EXEC to format a temporary OS disk */

signal on error
cp define t3340 198 10
cp close reader
cp purge reader class i
cp spool punch to '*' class i cont nohold
punch ipl dsf '* ('noh
punch temp dsf '* ('noh
cp spool punch nocont close
cp spool reader class i nohold
cp ipl 00c clear attn
exit

Error:
  exit 100
```

You execute this procedure by entering the filename of the EXEC:

```
tempdisk
```

When the final line of this EXEC is executed, the Device Support Facility is in control. You will receive the following three messages:

```
ICK005E DEFINE INPUT DEVICE, REPLY 'DDDD,CUU OR CONSOLE'
ENTER INPUT/COMMAND:
```

You should enter:

```
2540,00c
```

to indicate that the control statements should be read from your card reader, which is a virtual 2540 device at virtual address 00C.

```
ICK006E DEFINE OUTPUT DEVICE, REPLY 'DDDD,CUU OR CONSOLE'
ENTER INPUT/COMMAND:
```

You should enter:

```
console
```

to indicate that the utility output should sent to your console.

```
ICK003D REPLY U TO ALTER VOLUME 198 CONTENTS, ELSE T
ENTER INPUT/COMMAND:
```

You should enter:

```
u
```

to continue the execution.

When the Device Support Facilities program is completed, your virtual machine is in a wait state and you must reload CMS (with the IPL command). You can then access the temporary disk:

```
acc 198 c
```

and CMS responds:

```
C (198) R/W - DOS
```

Defining DOS Input and Output Files

Note: This information is for VSE/VSAM users. OS/VS VSAM users should refer to the section “Defining OS Input and Output Files”. You may use the DLBL command to define VSAM input and output files for both the AMSERV command and for program execution. The operands required on the DLBL command are:

```
dlbl ddname filemode DSN datasetname (options SYSxxx
```

where “ddname” corresponds to the FILE parameter in the AMSERV file and “datasetname” corresponds to the entry name or filename of the VSAM file.

Note: In the CMS/DOS and CMS/VSAM environments, filemodes "R" and "T" cannot be used on the DLBL command. These filemodes cannot be used because in CMS/DOS and CMS/VSAM, "R" and "T" are used as abbreviations for reader and terminal.

There are several options you can use when issuing the DLBL command to define VSAM input and output files. These options are:

VSAM indicates that the file is a VSAM file.

Note: You do not have to use the VSAM option to identify a file as a VSAM file if you are using any of the other options listed here, since they imply that the file is a VSAM file. In addition, the ddnames (filenames) IJSYSCT and IJSYSUC also indicate that the file being defined is a VSAM file.

EXTENT defines a catalog or a VSAM data space. You are prompted to enter the volume information. This option provides the function of the EXTENT card in VSE.

MULT accesses a multivolume VSAM file. You are prompted to enter the extent information.

CAT identifies a catalog that contains the entry for the VSAM file you are defining.

BUFSP specifies the size of the buffers VSAM should use during program execution.

Options are entered following the open parenthesis on the DLBL command line, with the SYSxxx:

```
assgn sys003 e
dlbl file1 b1 dsn workfile (extent cat cat2 sys003
```

Using VSAM Catalogs

While you are developing and testing your VSAM programs in CMS, you may find it convenient to create and use your own master catalog, which may be on a CMS minidisk. VSAM catalogs, like any other cluster, can be shared read-only among several users.

You name the VSAM master catalog for your terminal session using the logical unit SYSCAT in the ASSGN command and the ddname IJSYSCT for the DLBL command. For example, if your VSAM master catalog is located on a DOS disk you have accessed as a C-disk, you would enter:

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

Note: When you use the ddname IJSYSCT, you do not need to specify the VSAM option on the DLBL command.

You must define the master catalog at the start of every terminal session. If you are always using the same master catalog, you might include the ASSGN and DLBL commands in an EXEC procedure or in your PROFILE EXEC. You could also include the commands necessary to access the DOS system residence volume and enter the CMS/DOS environment:

```
ACCESS 350 Z
SET DOS ON Z (VSAM
ACCESS 555 C
ASSGN SYSCAT C
DLBL IJSYSCT C DSN MASTCAT (SYSCAT PERM
```

You should use the PERM option so that you do not have to reset the master catalog assignment after clearing previous DLBL definitions.

You must use the VSAM option on the SET DOS ON command if you want to use any access method services function or access VSAM files.

Defining a Master Catalog

The sample ASSGN and DLBL commands used above are almost identical to those you issue to define a master catalog using AMSERV. The only difference is the EXTENT option that lists the data spaces that this master catalog is to control.

As an example, suppose that you have a 30-cylinder 3330 minidisk assigned to you to use for testing your VSAM programs under CMS. Assuming that the minidisk is in your directory at address 333, you should first access it:

```
access 333 d
D (333) R/W - DOS
```

If you formatted the minidisk yourself, you know what its label is. If not, you can find out what the label is by using the CMS command:

```
query search
```

The response might be:

```
USR191  191  A    R/W
DOS333  333  D    R/W - DOS
SYS190  190  S    R/O
SYS19E  19E  Y/S  R/O
```

Use the label DOS333 in the VOLUMES parameter in the MASTCAT AMSERV file:

```
DEFINE MASTERCATALOG -
   (NAME  (MASTCAT ) -
    VOLUME (DOS333) -
    CYL (4) -
    FILE (IJSYSCT)    )
```

To find out what extents on the minidisk you can allocate for VSAM, use the LISTDS command with the FREE option:

```
listds d (free
```

The response from LISTDS might look like this:

```
FREESPACE INFORMATION FOR 'D' DISK:
CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
0000 01      1    0000 09      9        9
0000 11     11    0029 18    569      560
```

From this response, you can see that the volume table of contents (VTOC) is located on the first cylinder, so you can allocate cylinders 1 through 29 for VSAM:

```
assgn syscat d
dlbl ijsysct d dsn mastcat (syscat perm extent
DMSDLB331R Enter extent specifications:
19 551
   (null line)
```

After entering the extents, in tracks, giving the relative track number of the first track to be allocated followed by the number of tracks, you must enter a null line to complete the command. A null line is required because, when you enter multiple extents, entries may be placed on more than one line. If you do not enter a null line, the next line you enter causes an error, and you must re-enter all of the extent information.

Note: As in OS, the extents must be on cylinder boundaries, and you cannot allocate cylinder 0.

Now you can issue the AMSERV command:

```
amserv mastcat
```

A ready message with no return code indicates that the master catalog is defined. You do not need to reissue the ASSGN and DLBL commands in order to use the master catalog for additional AMSERV functions.

Defining User Catalogs

You can use the AMSERV command to define private catalogs and spaces for them. The procedures for determining what space you can allocate are the same as those outlined in the example of defining a master catalog.

To define a user catalog, you may use any programmer logical unit and any ddname:

```
access 199 e
listds e (free
.
.
.
assgn sys001 e
dlbl cat1 e dsn private.cat1 (sys001 extent perm
.
.
.
amserv usercat
```

The file USERCAT AMSERV might contain the following:

```
DEFINE USERCATALOG -
     (NAME (PRIVATE.CAT1) -
      CYL (4) -
      VOLUME (DOSVS2) ) -
      CATALOG (MASTCAT)
```

After this AMSERV command has completed successfully you can use the catalog PRIVATE.CAT1. When you issue a DLBL command to identify a cluster or data set cataloged in this catalog, you must identify the catalog using the CAT option on the DLBL command for the file:

```
assgn sys100 c
dlbl file2 c dsn ? (sys100 cat cat1
```

Or, you can define this catalog as a job catalog.

Using Job Catalogs

If you want to set up a user catalog as a job catalog so that it will be searched during all subsequent jobs, you can define the catalog using the special ddname IJSYSUC. For example:

```
assgn sys101 c
dlbl ijsysuc c dsn private.cat1 (sys101 perm
```

If you defined a user catalog (IJSYSUC) for a terminal session and you use the AMSERV command to access a VSAM file, the user catalog takes precedence over the master catalog. This means that for files that already exist, only the job catalog is searched. When you define a cluster, it is cataloged in the job catalog, rather than in the master catalog, unless you use the CAT option to override it.

If you want to use additional catalogs during a terminal session, you first define them just as you would any other VSAM file:

```
assgn sys010 f
dlbl mycat2 f dsn private.cat2 (sys010 vsam
```

Then, when you enter the DLBL command for the VSAM file that is cataloged in PRIVATE.CAT2, use the CAT option to refer to the ddname of the catalog:

```
assgn sys011 f
dlbl input f dsn input.file (sys011 cat mycat2
```

If you want to stop using a job catalog defined with the ddname IJSYSUC, you can clear it using the CLEAR option of the DLBL command:

```
dlbl ijsysuc clear
```

Then, the master catalog becomes the job catalog for files not defined with the CAT option.

Catalog Passwords

When you define passwords for VSAM catalogs in CMS, or when you use CMS to access VSAM catalogs that have passwords associated with them, you must supply the password from your terminal when the AMSERV command executes. The message you receive to prompt you for the password is the same message you receive when you execute access method services:

```
4221A ATTEMPT 1 OF 2. ENTER PASSWORD FOR JOB AMSERV
FILE catalog
```

When you enter the proper password, AMSERV continues execution.

Verifying A Catalog Structure

As a CMS VSAM user (with or without DOS set ON), you can use the CMS CATCHECK command to invoke the VSE/VSAM Catalog Check Service Aid to verify a complete catalog structure. If you do not specify a catalog name with the CATCHECK command, the job catalog specified with the DLBL command is used. CATCHECK produces a print file containing the catalog analysis. For example, issuing:

```
dlbl ijsysuc f dsn private.cat1 (vsam
```

and

```
catcheck
```

results in a print file containing the VSE/VSAM Catalog Check output.

If you had issued only a DLBL for the master catalog, issuing:

```
catcheck private.cat1
```

produces the same result.

Defining and Allocating Space for VSAM files

You can use CMS AMSERV to allocate additional data spaces for VSAM. To use the DEFINE SPACE control statement, you must have defined the catalog that will control the space, and you must have the volume or volumes where the space is to be allocated, mounted, and accessed.

For example, suppose you have a DOS-formatted 3330 disk attached to your virtual machine at virtual address 255. After accessing the disk and determining the free space on it, you could create a file named SPACE AMSERV:

```
DEFINE SPACE -
    (FILE (FILE1) -
     TRACKS (1900) -
     VOLUME (123456) ) -
     CATALOG (PRIVATE.CAT2 CAT2)
```

Before executing this AMSERV file, define PRIVATE.CAT2 as a user catalog using the ddname CAT2. Then define the ddname for the FILE parameter:

```
access 255 c
assgn sys010 c
dlbl cat2 c dsn private.cat2 (sys010 vsam
assgn sys011 c
dlbl file1 c (extent sys011 cat cat2
amserv space
```

You do not need to enter a data set name to define the space. When CMS prompts you for the extents of the space, you can enter the extent specifications:

```
DMSDLB331R Enter extent specifications:
190 1900
.
.
.
```

When you define space for VSAM, you should be sure that the VOLUMES parameter and the space allocation parameter (whether CYLINDER, TRACKS, BLOCKS, or RECORDS) in the AMSERV file agree with the information you provide in the DLBL command. All data extents must begin and end on cylinder boundaries. Any additional space you provide in the extent information that is beyond what you specified in the AMSERV file is claimed by VSAM.

Specifying Multiple Extents

When you are specifying extents for a master catalog, data space, or unique file, you can specify up to 16 extents on a volume for a particular space. When prompted by CMS to enter the extents, you must separate different extents by commas or place them on different lines. To specify a range of extents in the above example, you can enter:

```
dlbl file1 c (extent sys011
190  190, 570  190, 1900 1520
   (null line)
  -- or --
dlbl file1 c (extent sys011
190  190
570  190
1900 1520
   (null line)
```

Again, the first number entered for each extent represents the relative track for the beginning of the extent and the second number indicates the number of tracks.

Specifying Multivolume Extents

You can define spaces that span up to nine volumes for VSAM files. All of the volumes must be accessed and assigned when you issue the DLBL command to define or identify the data space.

You should remember, though, that if you are using AMSERV and you do not use the PRINT option, you must have a read/write CMS disk so that AMSERV can write the output LISTING file.

If you are defining a new multivolume data space or unique cluster, you must specify the extents on each volume that the data is to occupy (starting track and number of tracks) followed by the disk mode letter where the disk is accessed and the programmer logical unit to which the disk is assigned. For example:

```
access 135 b
access 136 c
access 137 d
assgn  sys001 b
assgn  sys002 c
assgn  sys003 d
dlbl newfile b (extent sys001
DMSDLB331R Enter extent specifications:
100 60 b sys001, 400 80 b sys001, 60 40 d sys003
2000 100 c sys002
   (null line)
```

If you specify more than one extent on the same line, the extents must be separated by commas. Different extents for the same volume must be entered consecutively.

Note: In the preceding example, the extent information is for 2314 disks. These extents are also on cylinder boundaries.

When you enter multivolume extents, you can use a default mode. For example:

```
dlbl newfile b (extent sys001
DMSDLB331R Enter extent specifications:
100 60, 400 80, 60 40 d sys003,
2000 100 c sys002
   (null line)
```

Any extents you enter without specifying a mode letter and SYSxxx value default to the mode and SYSxxx on the DLBL command line, in this case, the B-disk, SYS001.

If you make any errors issuing the DLBL command or extent information, you must re-enter the entire command sequence.

Identifying Existing Multivolume Files

When you issue a DLBL command to identify an existing multivolume VSAM file, you must use the MULT option of the DLBL command:

```
dlbl old b1 dsn ? (sys002 mult
DMSDLB220R Enter dataset name:
dostest.file
DMSDLB330R Enter volume specifications:
c sys004, d sys003
e sys007
  (null line)
```

When you enter the DLBL command, you should specify the mode letter and logical unit for the first volume on the command line. When you enter the MULT option, you are prompted to enter additional specifications for the remaining extents. In the preceding example, the data set has extents on disks accessed as B-, C-, D-, and E-disks.

Using Tape Input and Output

If you are using AMSERV for a function that requires tape input and/or output, you must have the tape(s) attached to your virtual machine. The valid addresses for tapes are 181 through 184. When referring to tapes, you can also refer to them using their CMS symbolic names TAP1 through TAP4.

For AMSERV functions that use tape input/output, the TLBL control statement is simulated by building a dummy DLBL containing a user-supplied ddname (filename). CMS does not read tape labels and does not recognize tape data set names.

When you invoke the AMSERV command, you must use the TAPIN or TAPOUT option to specify the tape device being used:

```
amserv export (tapout 181
```

In this example, the output from the AMSERV control statements in a file named EXPORT goes to a tape at virtual address 181. CMS prompts you to enter the ddname:

```
DMSAMS367R Enter tape output DDNAMEs:
```

After you enter the ddname specified on the FILE parameter in the AMSERV file and press the carriage return, the AMSERV command executes.

AMSERV opens all tape files as standard labelled tapes or non-labelled tapes. If you are using standard labelled tapes, you need to specify a LABELDEF command with AMSERV. The LABELDEF command is the CMS/DOS equivalent of VSE TLBL control statement. The LABELDEF command is used to specify information in VOL1 and HDR1 labels on the tape. See the description of the LABELDEF command in *VM/SP CMS Command Reference* for more information on this command.

You should use the same name for the filename on your LABELDEF command as you do for the ddname you enter in reply to message DMSAMS367R (the ddname specified on the FILE parameter in the AMSERV file). However, the LABELDEF command must be issued before the AMSERV command. The following sequence of commands might be used when you have standard labelled tape output:

```
assgn sys005 tap1
tape rew (181
assgn syscat e
assgn sys006 e
labeldef catout fid catfile volid amserv
dlbl ijsysct e dsn mastcat (syscat vsam
dlbl catin e dsn file (sys006 vsam
amserv repro (tapout 181
```

```
DMSAMS367R Enter tape output DDNAMEs:
```

```
catout
```

Note: If you do not care what is written in a tape output label or do not want input labels checked, you can specify a LABELDEF with no parameters other than filename. When you enter:

```
labeldef intape
```

for an input tape with ddname INTAPE, the standard labels on the tape are skipped without any checking. A similar statement for an output tape writes tape labels with default values (see the description of the LABELDEF command in *VM/SP CMS Command Reference.*)

If you use non-labelled tapes, LABELDEF is not required.

Reading VSAM Tape Files

When you create a tape in CMS using AMSERV, CMS writes a tape mark preceding each output file that it writes. When the same tape is read using AMSERV under CMS, HDR1 and VOL1 labels are checked using the LABELDEF command you provide. If you read this tape in a real VSE system, you should use a TLBL card instead of the LABELDEF command.

Similarly, when you create a tape under a VSE system using access method services, if the tape is created with standard labels, CMS AMSERV has no difficulty reading it.

The only time you should worry about positioning a tape created by AMSERV is when you want to read the tape using a method other than AMSERV, for example, the MOVEFILE command. Then, you must forward space the tape past the label using the CMS TAPE command before you can read it.

Defining OS Input and Output Files

Note: This information is for OS/VS VSAM users only. VSE/VSAM users should refer to "Defining DOS Input and Output Files" for information on defining files for use with VSAM.

The OS/VS VSAM user should bear in mind that CMS uses VSE/VSAM to manipulate VSAM files. The VSAM and AMS statements that can be used are described in the publication *Using VSE/VSAM Commands and Macros.*

In addition, there are certain incompatibilities between VSE/VSAM and OS/VS VSAM. For a description of these incompatibilities, refer to the *VSE/VSAM General Information Manual.*

If you are going to use access method services to manipulate VSAM or SAM files or you are going to execute VSAM programs under CMS, use the DLBL command to define the input and output files. The basic format of the DLBL command is:

```
DLBL ddname filemode DSN datasetname (options
```

where `ddname` corresponds to the FILE parameter in the AMSERV file and `datasetname` corresponds to the entry name of the VSAM file. That is, the name specified in the NAME parameter of an access method services control statement.

If you are using a CMS file for AMSERV input or output, use the CMS operand and enter CMS file identifiers as follows:

```
dlbl mine a cms out file1 (vsam
```

The maximum length allowed for ddnames under CMS VSAM is seven characters. This means that if you have assigned eight-character ddnames (or filenames) to files in your programs, only the first seven characters of each ddname are used. So, if a program refers to the ddname OUTPUTDD, you should issue the DLBL command for a ddname of OUTPUTD. Since you can encounter problems with a program that contains ddnames with the same first seven characters, you should recompile those programs using seven-character ddnames.

Note: In the CMS/DOS and CMS/VSAM environments, filemodes "R" and "T" cannot be used on the DLBL command. These filemodes cannot be used because in CMS/DOS and CMS/VSAM, "R" and "T" are used as abbreviations for reader and terminal.

There are several options you can use when issuing the DLBL command to define VSAM input and output files. These options are:

VSAM indicates that the file is a VSAM file.

Note: You do not have to use the VSAM option to identify a file as a VSAM file if you are using any of the other options listed here, since they imply that the file is a VSAM file. In addition, the ddnames (filenames) IJSYSCT and IJSYSUC also indicate that the file being defined is a VSAM file.

EXTENT defines a catalog or a VSAM data space. You are prompted to enter the volume information.

MULT accesses a multivolume VSAM file. You are prompted to enter the extent information.

CAT identifies a catalog which contains the entry for the VSAM file you are defining.

BUFSP specifies the size of the buffers VSAM should use during program execution.

Allocating Extents on OS Disks and Minidisks

When you use access method services to manipulate VSAM files under OS, you do not have to worry about allocating the real cylinders and tracks to contain the files. You can, however, use CMS commands to indicate which cylinders and tracks should contain particular VSAM spaces when you use the DEFINE control statement to define space.

Extents for VSAM data spaces can be defined, in AMSERV files, in terms of cylinders, tracks, or records. Extent information you supply to CMS when executing AMSERV must always be in terms of tracks. When you define data spaces or unique clusters, the extent information (number of cylinders, tracks, or records) in the AMSERV file must match the extents you supply when you issue the DLBL command to define the file. When you supply extent information for the master catalog, any extents you enter in excess of those required for the catalog are claimed by the catalog and used as data space.

CMS does not make secondary space allocation for VSAM data spaces. If you execute an AMSERV file that specifies a secondary space allocation, CMS ignores the parameter.

When you use the DLBL command to define VSAM data space, you can use the EXTENT option indicating to CMS that you are going to enter data extents. For example, if you enter:

```
dlbl space b (extent
```

CMS prompts you to enter the extents:

```
DMSDLB331R Enter extent specifications:
```

When you enter the extents, you specify the relative track number of the first track of the extent, followed by the number of tracks. For example, if you are allocating an entire 2314 disk, you would enter:

```
20 3980
   (null line)
```

You can never write on cylinder 0 track 0, and since VSAM data spaces must be allocated on cylinder boundaries, you should never allocate cylinder 0. Cylinder 0 is often used for the volume table of contents (VTOC) as well. Therefore, it is always best to begin defining space with cylinder 1.

You can determine what disk extents on an OS disk or minidisk are available for allocation by using the LISTDS command with the FREE option, which also indicates the relative track numbers as well as actual cylinder and head numbers.

Using VSAM Catalogs

While you are developing and testing your VSAM programs in CMS, you may find it convenient to create and use your own master catalog, which may be on a CMS minidisk. VSAM catalogs, like any other cluster, can be shared read-only among several users.

You name the VSAM master catalog for your terminal session using the ddname IJSYSCT for the DLBL command. For example, if your VSAM master catalog is located on an OS disk you have accessed as a C-disk, you would enter:

```
dlbl ijsysct c dsn master catalog (perm
```

You must define the master catalog at the start of every terminal session. If you are always using the same master catalog, you might include the DLBL command you need to define it in your PROFILE EXEC:

```
ACCESS 555 C
DLBL IJSYSCT C DSN MASTCAT (PERM
```

You should use the PERM option so that you do not have to reset the master catalog assignment after clearing previous DLBL definitions. The command:

```
dlbl * clear
```

clears all file definitions except those entered with the PERM option.

Defining a Master Catalog

The sample DLBL command used in the preceding example is almost identical with the one you would issue to define a master catalog using AMSERV. The only difference is that you can enter the EXTENT option so you can list the data spaces that this master catalog is to control.

As an example, suppose that you have a 30-cylinder 3330 minidisk assigned to you to use for testing your VSAM programs under CMS. Assuming that the minidisk is in your directory at address 333, you should first access it:

```
access 333 d
D (333) R/W - DOS
```

If you formatted the minidisk yourself, you know what label you assigned it. If not, you can find out the label assigned to the disk by issuing the CMS command:

```
query search
```

The response might be:

```
USR191  191  A    R/W
VSAM03  333  D    R/W - DOS
SYS109  190  S    R/O
SYS19E  19E  Y/S  R/O
```

Use the volume label VSAM03 in the MASTCAT AMSERV file:

```
 DEFINE MASTERCATALOG -
    (NAME  (MASTCAT ) -
     VOLUME (VSAM03) -
     CYL (4) -
     FILE (IJSYSCT)    )
```

To find out what extents on the minidisk you can allocate for VSAM, use the LISTDS command with the FREE option:

```
listds d (free
```

The response from LISTDS might look like this:

```
FREESPACE INFORMATION FOR 'D' DISK:
CYL-HD(RELTRK) TO CYL-HD(RELTRK)   TRACKS
0000 01     1   0000 09       9        9
0000 11    11   0029 18     569      560
```

From this response, you can see that the VTOC is located on the first cylinder, so you can allocate cylinders 1 through 29 for VSAM:

```
dlbl ijsysct d dsn mastcat (perm extent
DMSDLB331R Enter extent specifications:
19 551
    (null line)
```

After entering the extents, in tracks, giving the relative track number of the first track to be allocated followed by the number of tracks, you must enter a null line to complete the command. (A null line is required because, when you enter multiple extents, entries may be placed on more than one line.)

Now you can issue the AMSERV command:

```
amserv mastcat
```

A ready message with no return code indicates that the master catalog is defined. You do not need to reissue the DLBL command in order to identify the master catalog for additional AMSERV functions.

Defining User Catalogs

You can use the AMSERV command to define private catalogs and spaces for them. The procedures for determining what space you can allocate are the same as those outlined in the example of defining a master catalog.

To define a user catalog, you can assign any ddname you want:

```
access 199 e
listds e (free
.
.
.
dlbl cat1 e dsn private.cat1 (extent
.
.
.
amserv usercat
```

The file USERCAT AMSERV might contain the following:

```
DEFINE USERCATALOG -
     (NAME (PRIVATE.CAT1) -
      FILE  (CAT1) -
      CYL (4) -
      VOLUME (OSVSAM) ) -
      CATALOG (MASTCAT)
```

After this AMSERV command has completed successfully, you can use the catalog PRIVATE.CAT1. When you define a file cataloged in it, you identify the catalog using the CAT option on the DLBL command:

```
dlbl file2 e dsn ? (cat cat1
```

Or, you can define it as a job catalog.

Using a Job Catalog

During a terminal session, you may be referencing the same private catalog many times. If this is the case, you can identify a job catalog by using the ddname IJSYSUC. Then, that catalog is searched during all subsequent jobs unless you override it using the CAT option when you use the DLBL command to define a file.

If you defined a user catalog (IJSYSUC) for a terminal session and you use the AMSERV command to access a VSAM file, the user catalog takes precedence over the master catalog. This means that for files that already exist, the job catalog is searched. When you define a cluster, it is cataloged in the job catalog, rather than in the master catalog, unless you use the CAT option to override it. CMS never searches more than one VSAM catalog.

You should use the CAT option to name a catalog when the AMSERV file you are executing references, with the CATALOG parameter, a catalog that is not defined either as the master catalog or as a user catalog.

If you want to use additional catalogs during a terminal session, you first define them just as you would any other VSAM file:

```
dlbl mycat2 f dsn private.cat2 (vsam
```

Then, when you enter the DLBL command for the VSAM file that is cataloged in PRIVATE.CAT2 use the CAT option to refer to the ddname of the catalog:

```
dlbl input f dsn input.file (cat mycat2
```

If you want to stop using a job catalog defined with the ddname IJSYSUC, you can clear it using the CLEAR option of the DLBL command:

```
dlbl ijsysuc clear
```

or, you can assign the ddname IJSYSUC to some other catalog. If you clear the ddname for IJSYSUC, then the master catalog becomes the job catalog.

Catalog Passwords

When you define passwords for VSAM catalogs in CMS or when you use CMS to access VSAM catalogs that have passwords associated with them, you must supply the password from your terminal when the AMSERV command executes. The message you receive to prompt you for the password is the same message you receive when you execute access method services:

```
4221A ATTEMPT 1 OF 2. ENTER PASSWORD FOR JOB AMSERV      FILE catalog
```

When you enter the proper password, AMSERV continues execution.

Verifying a Catalog Structure

As a CMS VSAM user (with or without DOS set ON), you can use the CMS CATCHECK command to invoke the VSE/VSAM Catalog Check Service Aid to verify a complete catalog structure. If you do not specify a catalog name with the CATCHECK command, the catalog specified with the DLBL command is used. CATCHECK produces a print file containing the catalog analysis. For example, issuing:

```
dlbl ijsysuc f dsn private.cat1 (vsam
```

and

```
catcheck
```

results in a print file containing the VSE/VSAM Catalog Check output.

If you had issued only a DLBL for the master catalog, issuing:

```
catcheck private.cat1
```

produces the same result.

Defining and Allocating Space for VSAM files

You can use CMS AMSERV to allocate additional data spaces for VSAM. To use the DEFINE SPACE control statement, you must have defined either the master catalog or a user catalog that will control the space, and you must have the volume or volumes where the space is to be allocated, mounted, and accessed.

For example, suppose you have an OS 3330 disk attached to your virtual machine at virtual address 255. After accessing the disk and determining the free space on it, you could create a file named SPACE AMSERV:

```
DEFINE SPACE -
   (FILE (FILE1) -
    TRACKS (1900) -
    VOLUME (123456) ) -
    CATALOG (PRIVATE.CAT2 CAT2)
```

Before executing this AMSERV file, define PRIVATE.CAT2 using the ddname CAT2. Then define the ddname for the file:

```
access 255 c
dlbl cat2 c dsn private.cat2 (vsam
dlbl file1 c (extent cat cat2
```

You do not need to enter a data set name to define the space. When CMS prompts you for the extents of the space, you can enter the extent specifications:

```
DMSDLB331R Enter extent specifications:
190 1900
.
.
.
```

When you define space for VSAM, you should be sure that the VOLUMES parameter and the space allocation parameter (whether CYLINDER, TRACKS, BLOCKS, or RECORDS) in the AMSERV file agree with the track information you provide in the DLBL command.

Specifying Multiple Extents

When you are specifying extents for a master catalog, data space, or unique file, you can specify up to 16 extents on a volume for a particular space. When prompted by CMS to enter the extents, you must separate the different extents by commas or place them on different lines. To specify a range of extents in the above example, you could enter:

```
dlbl file1 c (extent
190  190, 570  190, 1900 1520
   (null line)
  -- or --
dlbl file1 c (extent
190  190
570  190
1900 1520
   (null line)
```

Again, the first number entered for each extent represents the relative track for the beginning of the extent and the second number indicates the number of tracks.

Specifying Multivolume Extents

You can define spaces that span up to nine volumes for VSAM files. All of the volumes must be accessed and assigned when you issue the DLBL command to define or identify the data space.

You should remember, though, that if you are using AMSERV and you do not use the PRINT option, you must have a read/write CMS disk so that AMSERV can write the output LISTING file.

If you are defining a new multivolume data space or unique cluster, you must specify the extents on each volume that the data is to occupy (starting track and number of tracks) followed by the disk mode letter at which the disk is assigned:

```
access 135 b
access 136 c
access 137 d
dlbl newfile b (extent
DMSDLB331R Enter extent specifications:
100 60 b, 400 80 b, 60 40 d,
2000 100 c
   (null line)
```

If you specify more than one extent on the same line, the extents must be separated by commas. If you enter a comma at the end of a line, it is ignored. Different extents for the same volume must be entered consecutively.

Note: In this example, the extent information is for 2314 disks. These extents are also on cylinder boundaries.

When you enter multivolume extents, you do not have to enter a mode letter for those extents on the disk identified in the DLBL command. For the extents on disk B in the above example, you could enter:

```
dlbl newfile b (extent
DMSDLB331R Enter extent specifications:
100 60, 400 80, 60 40 d
2000 100 c
   (null line)
```

If you make any errors issuing the DLBL command or extent information, you must reissue the entire command sequence.

Identifying Existing Multivolume Files

When you issue a DLBL command to identify an existing multivolume VSAM file, you must use the MULT option of the DLBL command:

```
dlbl old b1 dsn ? (mult
DMSDLB220R Enter dataset name:
vsamtest.file
DMSDLB330R Enter volume specifications:
c, d
e
   (null line)
```

When you enter the DLBL command you should specify the mode letter for the first disk volume on the command line. When you enter the MULT option you are prompted to enter additional specifications for the remaining extents. In the above example, the data set has extents on disks accessed as B-, C-, D-, and E-disks.

Using Tape Input and Output

If you are using AMSERV for a function that requires tape input and/or output, you must have the tape(s) attached to your virtual machine. The valid addresses for tapes are 181 through 184. When referring to tapes, you can also refer to them using their CMS symbolic names TAP1 through TAP4.

When you use AMSERV to create or read a tape, you supply the ddname for the tape device interactively, after you issue the AMSERV command. To indicate to AMSERV that you are using tape for input or output, you must use the TAPIN or TAPOUT option to specify the tape device being used:

```
labeldef tapedd fid filename...
amserv export (tapout 181
```

In this example, the output from an EXPORT function is to a tape at virtual address 181. CMS prompts you to enter the ddname:

```
DMSAMS367R Enter tape output DDNAMEs:
```

After you enter the ddname (TAPEDD in this example) for the tape file, AMSERV begins execution.

AMSERV in CMS assumes that tape volumes used for input and/or output have IBM standard tape labels, i.e., VOL1, HDR1, etc. The user can override this default by indicating to AMSERV via Access Method Services control statements to use nonlabel tapes. If standard label tapes are used, the LABELDEF command is required. The CMS/DOS routine that performs the tape open needs label information for standard label tapes. See the description of the LABELDEF command in the *VM/SP CMS Command Reference* for further information. The filename you specify on the LABELDEF command should be the same one you use to reply to the access method service message that requested you to supply the tape's ddnames. However, the LABELDEF command must be issued before the AMSERV command. If you only want the tape labels skipped, but not checked, enter a LABELDEF with no parameters other than filename.

Standard label tapes used for input must always contain standard VOL1, HDR1, and EOF1 labels or they are rejected by CMS AMSERV. Standard label output tapes do not need to contain VOL1 labels because the user is prompted to enter a volume serial number and have the VOL1 label written if he wants. However, blank tapes should not be used for output because the open routine tries to read the tape.

Reading Tapes

When you create a tape file using AMSERV under CMS, CMS writes a label file preceding each output file. When CMS AMSERV is used to read this same file, it checks the HDR1 and VOL1 labels using the LABELDEF command you provide before it reads the data file. If you want to read the tape on a real OS/VS system, however, you must use either LABEL=SL or

LABEL=(2,NL) as a parameter on the data definition (DD) card for the tape.

If you are creating a tape under OS/VS access method services to be read by CMS AMSERV, you must be sure to create the tape using standard labels so that CMS can read it properly. CMS cannot read a tape created with LABEL=(,NL) on the DD card.

For CMS to read this tape for any other purpose (for example, to use the MOVEFILE command to copy it), you must remember to forward space the file past the tape mark before beginning to read it.

Using AMSERV under CMS

This section provides examples of AMSERV functions executed under CMS. The examples are applicable to both the CMS (OS) and CMS/DOS environments. You should be familiar with the material presented in either "Defining DOS Input and Output Files" or "Defining OS Input and Output Files," depending on whether you are a DOS or an OS user, respectively. For the examples shown below, command lines and options that are required only for CMS/DOS users are shaded. OS users should ignore these shaded entries.

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

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

A CMS standard variable-format file can be accessed as RECFM=U to use the file as follows:

```
AMSERV   AMREPUV
```

The file AMREPUV AMSERV contains the following 2 cards:

```
REPRO   INFILE (INPUT ENV(RECFM(U),BLKSZ(800),PDEV(3330)))
        OUTFILE (OUTPUT ENV(RECFM(V),BLKSZ(800),RECSZ(84),PDEV(3330)))
```

The input file can be any CMS file with LRECL 800 or less. The output file will be a true variable file that can be used with AMSERV.

The DEFINE and DELETE Functions

When you use the DEFINE and DELETE control statements of AMSERV, you do not need to specify the DSN parameter on the DLBL command:

```
assgn syscat c
dlbl ijsysct c (perm extent syscat
```

If the above commands are executed prior to an AMSERV command to define a master catalog, the DEFINE will be successful as long as you have assigned a data set name using the NAME parameter in the AMSERV file. The same is true when you define clusters or when you use the DELETE function to delete a cluster, space, or catalog.

When you do not specify a data set name, AMSERV obtains the name from the AMSERV file. In the case of defining or deleting space, no data set name is needed. The FILE parameter corresponding to the ddname is all that is necessary, and AMSERV assigns a default data set name to the space.

When you define space on a minidisk using AMSERV, CMS does not check the extents you specify to see whether they are greater than the number of cylinders available. As long as the starting cylinder is a valid cylinder number and the extents you specify are on cylinder boundaries, the DEFINE function completes successfully. However, you receive an error message when you use an AMSERV function that tries to use this space.

Defining a Suballocated Cluster

To define a cluster for VSAM space that has already been allocated, you need:

1. An AMSERV file containing the control statements necessary for defining the cluster, and

2. The master catalog (and, perhaps, user catalog) volume, which will point to the cluster.

The volume where the cluster is to reside does not have to be online when you define a suballocated cluster.

For example, the file CLUSTER AMSERV contains the following:

```
DEFINE CLUSTER  (  NAME (BOOK.LIST) -
      VOLUMES (123456) -
      TRACKS (40) -
      KEYS (14,0) RECORDSIZE (120,132)  ) -
    DATA (NAME (BOOK.LIST.DATA) ) -
    INDEX (NAME (BOOK.LIST.INDEX) )
```

To execute this file, you would need to enter the following command sequence (assuming that the master catalog, on volume 123456, is in your virtual machine at address 310):

```
access 310 b
assgn syscat b
dlbl ijsysct b (perm syscat
amserv cluster
```

Defining a Unique Cluster

For a unique cluster (one defined with the UNIQUE attribute), you must define the space for the cluster at the same time you define its name and attributes. Therefore, the volume or volumes where the cluster is to reside must be mounted and accessed when you execute the AMSERV command. You can supply extent information for the cluster's data and index portions separately.

Suppose UNIQUE AMSERV contains the following (the ellipses indicate that the AMSERV file is not complete):

```
DEFINE CLUSTER -
      (NAME (PAYROLL) ) -
      DATA ( FILE (UDATA) -
             UNIQUE -
             VOLUMES (567890) -
             CYLINDERS (40) -
             ... ) -
      INDEX ( FILE (UINDEX) ) -
             UNIQUE -
             VOLUMES (567890) -
             CYLINDERS (10) -
             ... )
```

To execute UNIQUE AMSERV, issue the following command sequence:

```
access 350 c
assgn sys004 c
dlbl udata c (extent sys004
DMSDLB331R Enter extent specifications:
800 800 c sys004
dlbl uindex c (extent sys004
600 200 c sys004
amserv unique
```

Deleting Clusters, Spaces, and Catalogs

When you use AMSERV to delete a VSAM cluster, the volume containing the cluster does not have to be accessed unless the volume also contains the catalog where the cluster is defined. In the case of data spaces and user catalogs or the master catalog, the volume(s) must be mounted and accessed in order to delete the space.

When you delete a cluster or a catalog, you do not need to use the DLBL command, except to define the master catalog; AMSERV can obtain the necessary file information from the AMSERV file.

When you are using temporary disks with AMSERV, you should be particularly careful that you have not cataloged a temporary data space or cluster in a permanent catalog. You will not be able to delete the space or cluster from the catalog.

The REPRO, IMPORT, and EXPORT (or EXPORTRA/IMPORTRA) Functions

You can manipulate VSAM files in CMS with the REPRO, IMPORT, and EXPORT functions of AMSERV. You can create VSAM files from sequential tape or disk files (on OS, DOS, or CMS disks) using the REPRO function. Using REPRO, you can also copy VSAM files into CMS disk files or onto tapes. For the IMPORT/EXPORT process, you have the option (for smaller files) of exporting VSAM files to CMS disks or to tapes.

You cannot, however, use the EXPORT function to write files onto OS or DOS disks. Nor can you use the REPRO function to copy ISAM (indexed sequential) files into VSAM data sets, since CMS cannot read ISAM files.

When creating a VSAM file from a non-VSAM disk file, the device track size must be the maximum BLOCKSIZE in the INFILE statement. AMSERV expects a DOS or OS file as input and will not open a disk file when the BLOCKSIZE specified is greater than the track capacity of the disk device being used.

You cannot use the ERASE or PURGE options of the EXPORT command if you are exporting a VSAM file from a read-only disk. The export operation succeeds, but the listing indicates an error code 184, meaning that the erase function could not be performed.

You should not use an EXPORT DISCONNECT function from a CMS minidisk and try to perform an IMPORT CONNECT function for that data set onto an OS system. OS incorrectly rebuilds the data set control block (DSCB) that indicates how much space is available.

Copying a CMS Sequential File into a VSAM File

The AMSERV file below gives an example of using the REPRO function to copy a CMS sequential file into a VSAM file. The CMS input file must be sorted in alphameric sequence before it can be copied into the VSAM file, which is a keyed sequential data set (KSDS). The VSAM cluster, NAME.LIST, is defined in an AMSERV file named PAYROLL:

```
DEFINE CLUSTER ( NAME (NAME.LIST   ) -
       VOLUMES (CMSDEV) -
       TRACKS (20) -
       KEYS (14,0) -
       RECORDSIZE (120,132) ) -
    DATA (NAME (NAME.LIST.DATA) ) -
    INDEX (NAME (NAME.LIST.INDEX ) )
```

To sort the CMS file, create the cluster, and copy the CMS file into it, use the following commands:

```
sort name list a name sort a
DMSSRT604R Enter sort fields:
1 14
access 135 c
assgn syscat c
dlbl ijsysct c (perm syscat
amserv payroll
assgn sys006 a
dlbl sort a cms name sort (sys006
assgn sys007 c
dlbl name c dsn name list (sys007 vsam
amserv repro
```

The file REPRO AMSERV contains:

```
REPRO   INFILE ( SORT  -
          ENV (RECORDFORMAT (F) -
               BLOCKSIZE (80) -
               PDEV (3330) ) ) -
         OUTFILE (NAME)
```

EXPORTing a VSAM File to a Tape

When you use the REPRO, IMPORT, or EXPORT functions with tape files, you must remember to use the TAPIN and TAPOUT options of the AMSERV command. These options perform two functions:

- they allow you to specify the device address of the tape
- they notify AMSERV to prompt you to enter a ddname

In the example below, a VSAM file is being exported to a tape. The file, TEXPORT AMSERV, contains:

```
EXPORT  NAME.LIST -
        INFILE (NAME) -
        OUTFILE (TAPE ENV (PDEV (2400) ) )
```

To execute this AMSERV, you enter the commands as follows:

```
assgn sys006 c
dlbl name c (sys006 vsam
amserv texport (tapout 181
DMSAMS367R Enter tape output DDNAMEs:
tape
```

The `fid`, `volid`, and `exdte` parameters on LABELDEF are only examples. You can substitute any value you want for them on your tape label.

Writing EXECs for AMSERV and VSAM

You may find it convenient to use EXEC procedures for most of your AMSERV functions, as well as setting up input and output files for program execution, and executing your VSAM programs. If, for example, a particular AMSERV function requires several disks and a number of DLBL statements, you can place all of the required commands in an EXEC file.

Suppose you have the following file called SETUP EXEC:

```
/*   */
ACCESS 135 B
ACCESS 136 C
ACCESS 137 D
ACCESS 300 G
ASSGN SYSCAT G
DLBL IJSYSCT G '('PERM SYSCAT
ASSGN SYS001 B
DLBL FILE1 B DSN FIRST FILE '('VSAM SYS001
ASSGN SYS002 C
DLBL FILE2 C DSN SECOND FILE '('VSAM SYS002
ASSGN SYS003 D
DLBL FILE3 D DSN THIRD FILE '('VSAM SYS003
AMSERV MULTFILE
```

To invoke this sequence of commands, enter the name of the EXEC:

```
setup
```

If you place the following statement at the beginning of the EXEC file:

```
signal on error
```

and then place the following statements at the end of the EXEC:

```
Error:
Say 'Unexpected return code' rc 'from command'
Say sourceline(sigl) 'at line' sigl'.'
Exit rc
```

you can be sure that the AMSERV command does not execute unless all of the prior commands completed successfully.

For those AMSERV functions that issue response messages, you can use the REXX PUSH or QUEUE instructions. For example:

```
/* An exec to invoke AMSERV */
signal on error
access 305 d
assgn sys007 d
dlbl output d '('vsam sys007
labeldef tape fid file1
signal off error
push tape
amserv timport '('tapin 181
if rc ¬= 0
  then type timport listing
tape rew
exit 0

Error:
Say 'Unexpected return code' rc 'from command'
Say sourceline(sigl) 'at line' sigl'.'
Exit rc
```

When the AMSERV command in the EXEC is executed, the request for the tape ddname is satisfied immediately by the response stacked with the PUSH statement.

If you are executing a command that accepts multiple response lines, you have to stack a null line as follows:

```
queue c sys002',' d sys003
queue
dlbl multfile b '('mult sys001
```

ISAM Interface Program (IIP)

CMS does not support the VSAM ISAM Interface Program (IIP). Thus, any program that creates and accesses ISAM (indexed sequential access method) data sets cannot be used to access VSAM key sequential data sets. There is one exception to this restriction. You can execute VSAM I/O requests if:

1. Your OS PL/I programs have files declared as ENV(INDEXED), and
2. The library routines detect that the data set being accessed is a VSAM data set.

VSE/VSAM Macros Supported

The VSE/VSAM macros and their options are supported for use in assembler language programs under CMS/DOS. The VSE/VSAM macros are:

ACB	EXLST	SHOWCAT
BLDVRP	GENCB	SHOWCB
DLVRP	MODCB	TCLOSE
ENDREQ	POINT	TESTCB
ERASE	RPL	WRTBFR

All options are supported with the exception of "AM=VTAM". This option is not supported on any of the macros.

The EXLST EXCPAD exit may be specified, but it is never taken in the CMS environment. The reason is that VSE/VSAM takes this exit when it is waiting for I/O to complete, but in the CMS environment, I/O is always complete when control is returned to VSE/VSAM.

In addition to the above list of macros, the following list of VSE macros normally used with the VSAM macros are also supported. The following macros are distributed with CMS for use with VSAM only.

VSE Macro Supported	Extent of Support
CDLOAD	Only supported to the extent required for VSAM execution.
CLOSE	Supported for both VSAM and SAM.
CLOSER	Supported for both VSAM and SAM.
GET	Supported for both VSAM and SAM.
OPEN	Supported for both VSAM and SAM.
OPENR	Supported for both VSAM and SAM.
PUT	Supported for both VSAM and SAM.

Obtaining the VSE/VSAM Macros

The "VSEVSAM EXEC" obtains the VSE/VSAM assembler language macros from the VSE/VSAM Licensed Optional Machine Readable Materials tape. The "VSESVAM EXEC" then creates a VSE/VSAM MACLIB. To install the VSE/VSAM assembler language macros, do the following:

1. Mount the Licensed Optional Machine Readable Materials tape on virtual 181.

2. Load the seven VSE macros (CDLOAD, CLOSE, CLOSER, GET, OPEN, OPENR, and PUT) from the product tape to disk (MAINT 393 is

recommended). The actual disk used is not important as long as the macros are available when VSEVSAM is issued.

3. Issue the CMS VSEVSAM command. Respond to the questions when you are prompted.

The seven VSE macros can be erased from the disk after the MACLIB is created because the macros will be in the MACLIB. Once you have created the MACLIB, you are able to assemble your VSAM assembler applications using the VSE/VSAM assembler macros in the MACLIB. The VSEVSAM EXEC is documented in the *VM/SP Installation Guide.*

VSE Supervisor Macros and Logical Transients Support for VSAM

VSE supervisor macros required by VSE/VSAM are supported by CMS. See Figure 25 on page 237 for a complete list of supervisor macros supported.

CMS distributes the VSE transients that are needed in the VSAM support. Thus, OS users do not need to have the VSE system pack on-line when they are compiling and executing VSAM programs.

CMS uses all of the VSE B-transients except those that build and release extent blocks. The extent block is not supported in CMS and, thus, neither are the B-transients that control extent blocks.

The CMSDOS shared segment contains the B-transients that are simulated for VSE support in CMS. The B-transients pertaining only to VSAM are included in the VSAM saved segment. Other VSE routines required by VSE/VSAM are contained in the CMSBAM shared segment. This includes the common VTOC handler routines, SAM data management, and the VSAM look-aside function.

OS/VSAM Macros Supported in CMS

A subset of the OS/VSAM macros are supported for use in CMS. The macros are at an MVS 3.8 level and they are contained in the OSVSAM MACLIB that is shipped with VM/SP. The macros are:

ACB	EXLST	RPL
CHECK	GENCB	SHOWCB
ENDREQ	MODCB	TESTCB
ERASE	POINT	

Some options of the OS/VSAM macros do not work in CMS because OS/VSAM macro requests are executed using VSE/VSAM code. Figure 34 lists the OS/VSAM macros and the supported options.

OS/VSAM Macro	Supported Options	

ACB	AM=VSAM
	BUFND=number
	BUFNI=number
	BUFSP=number
	DDNAME=ddname
	MACRF=ADR, CNV, KEY, NDF,
	DIR, SEQ, SKP, IN, OUT,
	NRM|AIX, NRS|RST, NSR, NUB|UBF,
		EXLST=address
		MAREA=address
		MLEN=number
		PASSWD=address
		STRNO=number

CHECK	RPL=address	
ENDREQ	RPL=address	
ERASE	RPL=address	

EXLST	AM=VSAM
	EODAD=address
	JRNAD=address
	LERAD=address
	SYNAD=address	

GENCB BLK=ACB	EXLST=address
	BUFND=number
	BUFNI=number
	BUFSP=number
	COPIES=number
	DDNAME=ddname
	MACRF=ADR, CNV, KEY, NDF,
	DIR, SEQ, SKP, IN, OUT,
	NRM|AIX, NRS|RST, NSR, NUB|UBF,
		LENGTH=number
		MAREA=address
		MLEN=number
		PASSWD=address
		STRNO=number
		WAREA=address

GENCB BLK=EXLST	EODAD=address
	JRNAD=address
	LERAD=address
	SYNAD=address
		COPIES=number
		LENGTH=number
		WAREA=address

GENCB BLK=RPL	ACB=address
	AREA=address
	AREALEN=number
	ARG=address
	COPIES=number
	OPTCD=ADR|CNV|KEY,
	DIR|SEQ|SKP, ARD|LRD,
	FWD|BWD, ASY|SYN,
	NSP|NUP|UPD, KEQ|KGE,
	FKS|GEN, LOC|MVE,
		ECB=address
		KEYLEN=number
		LENGTH=number
		NXTRPL=address
		RECLEN=number
		WAREA=number

Figure 34 (Part 1 of 3). Options of OS/VSAM Macros Supported in CMS

OS/VSAM Macro	Supported Options	
GET	RPL = address	
MODCB ACB	BUFND = number	EXLST = address
	BUFNI = number	MAREA = address
	BUFSP = number	MLEN = address
	DDNAME = ddname	PASSWD = address
	MACRF = ADR, CNV, KEY, NDF,	STRNO = number
	DIR, SEQ, SKP, IN, OUT,	
	NRM|AIX, NRS|RST, NSR,	
	NUB|UBF,	
MODCB EXLST	EODAD = address	
	JRNAD = address	
	LERAD = address	
	SYNAD = address	
MODCB RPL	ACB = address	ECB = address
	AREA = address	KEYLEN = number
	AREALEN = number	NXTRPL = address
	ARG = address	RECLEN = number
	OPTCD = ADR|CNV|KEY,	
	DIR|SEQ|SKP, ARD|LRD,	
	FWD|BWD, ASY|SYN,	
	NSP|NUP|UPD, KEQ|KGE,	
	FKS|GEN, LOC|MVE	
POINT	RPL = address	
RPL	ACB = address	ARG = address
	AM = VSAM	ECB = address
	AREA = address	KEYLEN = number
	AREALEN = number	NXTRPL = address
	OPTCD = ADR|CNV|KEY,	RECLEN = number
	DIR|SEQ|SKP, ARD|LRD,	
	FWD|BWD, ASY|SYN,	
	NSP|NUP|UPD, KEQ|KGE,	
	FKS|GEN, LOC|MVE,	
SHOWCB ACB	AREA = address	OBJECT = DATA|INDEX
	FIELDS = ACBLEN, AVSPAC,	LENGTH = number
	BUFND, BUFNI, BUFNO,	
	BUFSP, CINV, DDNAME,	
	ERROR, EXLST, FS, KEYLEN,	
	LRECL, MAREA, MLEN, NCIS,	
	NDELR, NEXCP, NEXT, NINSR,	
	NIXL, NLOGR, NRETR, NSSS,	
	NUPDR, PASSWD, RKP, STMST,	
	STRMAX, STRNO	
SHOWCB EXLST	AREA = address	LENGTH = number

Figure 34 (Part 2 of 3). Options of OS/VSAM Macros Supported in CMS

OS/VSAM Macro	Supported Options	
	FIELDS=EODAD, EXLLEN, JRNAD, LERAD, SYNAD	
SHOWCB RPL	AREA=address	LENGTH=number
	FIELDS=ACB, AIXPC, AREA, AREALEN, ARG, ECB, FDBK, FTNCD, KEYLEN, NXTRPL, RBA, RECLEN, RPLLEN	
TESTCB ACB	ERET=address	LRECL=number
	OBJECT=DATA|INDEX	MAREA=address
	ATRB=UNQ	MLEN=number
	OFLAGS=OPEN	NCIS=number
	OPENOBJ=PATH|BASE|AIX	NDELR=number
	ACBLEN=number	NEXCP=number
	AVSPAC=number	NEXT=number
	BUFND=number	NINSR=number
	BUFNI=number	NIXL=number
	BUFNO=number	NLOGR=number
	BUFSP=number	NRETR=number
	CINV=number	NSSS=number
	DDNAME=ddname	NUPDR=number
	ERROR=number	PASSWD=address
	EXLST=address	RKP=number
	FS=number	STMST=address
	KEYLEN=number	STRNO=number
	ATRB=ESDS, KSDS, REPL, RRDS, SPAN, SSWD, WCK	MACRF=ADR, AIX, CNV, DIR, IN, KEY, NDF, NRM, NRS, NSR, NUB, OUT, RST, SEQ, SKP, UBF
TESTCB EXLST	ERET=address	LERAD=address
	EODAD=address	SYNAD=address
	JRNAD=address	EXLLEN=number
TESTCB RPL	ERET=address	ARG=address
	AIXFLAG=AIXPKP	ECB=address
	AIXPC=number	FDBK=number
	FTNCD=number	KEYLEN=number
	I/O=COMPLETE	NXTRPL=address
	ACB=address	RBA=number
	AREA=address	RECLEN=number
	AREALEN=number	RPLLEN=number
	OPTCD=ADR, ARD, ASY, BWD, CNV, DIR, FKS, FWD, GEN, KEQ, KEY, KGE, LOC, LRD, MVE, NSP, NUP, SEQ, SKP, SYN, UPD	

Figure 34 (Part 3 of 3). Options of OS/VSAM Macros Supported in CMS

OS/VSAM Error Codes

Error codes returned by VSE/VSAM in response to OPEN, CLOSE, and Data Management Request macro errors are mapped to the appropriate OS/VSAM error codes.

- Figure 35 lists the error codes returned by VSE/VSAM in response to OPEN errors.
- Figure 36 on page 318 lists the error codes returned by VSE/VSAM in response to CLOSE errors.
- Figure 37 on page 319 lists the error codes returned by VSE/VSAM in response to Data Management Request macro errors.

If a VSE/VSAM error code cannot be mapped to any OS/VSAM error code, a CMS error message and an ABEND 35 are issued except for the cases indicated by an "*".

The following table lists the VSE/VSAM to OS/VSAM error code mapping for OPEN errors:

VSE/VSAM Error Code	CMS Error Message or OS/VSAM Error Code	VSE/VSAM Return Code	OS/VSAM Return Code
2	DMSVIP779E	8	8
4	4	8	8
14	DMSVIP782E	8	8
15	DMSVIP782E	8	8
17	DMSVIP782E	8	8
18	DMSVIP782E	8	8
19	DMSVIP782E	8	8
32	DMSVIP782E	8	8
34	DMSVIP782E*	8	8
40	DMSVIP778E	8	8
48	168	8	8
50	DMSVIP782E	8	8
64	188	8	8
65	DMSVIP779E	8	8
66	DMSVIP782E	8	8
67	DMSVIP782E	8	8

Figure 35 (Part 1 of 3). VSE/VSAM to OS/VSAM Error and Return Code Mapping for OPEN Errors

VSE/VSAM Error Code	CMS Error Message or OS/VSAM Error Code	VSE/VSAM Return Code	OS/VSAM Return Code
68	168	8	8
69	DMSVIP782E	8	8
70	DMSVIP782E	8	8
71	DMSVIP782E	8	8
72	148	8	8
78	DMSVIP782E	8	8
79	DMSVIP782E	8	8
80	DMSVIP778E	8	8
92	DMSVIP779E	8	8
96	96	4	4
100	100	4	4
104	104	4	4
108	108	4	4
110	160	8	8
113	144	0	4
114	DMSVIP781E	0	4
115	DMSVIP781E	8	8
116	116	4	4
117	DMSVIP782E	8	8
118	0	0	0
128	128	8	8
132	132	8	8
136	136	8	8
144	144	8	8
148	148	8	8
152	152	8	8
160	160	8	8
161	160	8	8
165	DMSVIP782E	8	8
166	DMSVIP782E	8	8
167	DMSVIP782E	8	8
168	168	8	8
180	180	8	8
188	DMSVIP782E	8	8

Figure 35 (Part 2 of 3). VSE/VSAM to OS/VSAM Error and Return Code Mapping for OPEN Errors

VSE/VSAM Error Code	CMS Error Message or OS/VSAM Error Code	VSE/VSAM Return Code	OS/VSAM Return Code
192	192	8	8
196	196	8	8
212	212	8	8
216	216	8	8
220	220	8	8
228	228	8	8
232	232	8	8
248	DMSVIP782E	8	8
254	DMSVIP782E	8	8
255	144	8	8

Figure 35 (Part 3 of 3). VSE/VSAM to OS/VSAM Error and Return Code Mapping for OPEN Errors

The following table lists the VSE/VSAM to OS/VSAM error code mapping for CLOSE errors:

VSE/VSAM Error Code	CMS Error Message or OS/VSAM Error Code	VSE/VSAM Return Code	OS/VSAM Return Code
2	DMSVIP783E	non-zero	4
4	4	non-zero	4
76	DMSVIP784	non-zero	4
136	136	non-zero	4
144	144	non-zero	4
165	DMSVIP784	non-zero	4
166	DMSVIP784	non-zero	4
167	DMSVIP784	non-zero	4
184	184	non-zero	4
188	0	non-zero	4
228	DMSVIP783	non-zero	4
252	DMSVIP784	non-zero	4
254	DMSVIP784	non-zero	4
255	148	non-zero	4

Figure 36. VSE/VSAM to OS/VSAM Error and Return Code Mapping for CLOSE Errors

For Data Management Request errors, all VSE/VSAM error codes are returned to the OS/VSAM user since the VSE/VSAM and OS/VSAM error codes are equivalent, with the following exceptions:

VSE/VSAM Error Code	CMS Error Message or OS/VSAM Error Code	VSE/VSAM Return Code	OS/VSAM Return Code
32	DMSVIP785E	0	0
48	40	8	8
52	Abend 52*	8	8
56	Abend 56*	8	8
128	DMSVIP786E	8	8
208	DMSVIP786E	8	8
212	DMSVIP786E	8	8
216	DMSVIP785E	8	8

Figure 37. DATA Management Request Error Return Code Mapping

Hardware Devices Supported

CMS support of VSAM data sets is based on VSE/VSAM. In its support of VSAM data sets, CMS uses RPS (rotational position sensing) wherever possible. CMS does not use RPS for 2314/2319 devices or for 3340 devices that do not have the feature. Except for the 3380, only disks supported by VSE can be used for VSAM data sets in CMS. These disks 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.

Saving the CMS System

Only named systems can be saved. The NAMESYS macro must be used to name a system. A discussion on creating a named system is found in the *VM/SP Planning Guide and Reference.*

The DMKSNT file must have been configured (by coding the NAMESYS macro) when CP was generated. The DMKSNT file contains the system name, size of the system, and its real disk location. The CMS system may be saved by:

- Providing a positive response to the following message (message 729R):

  ```
  Do you want to save the system? (enter 1=YES or 0=NO)
  ```

- Modifying the DEFNUC MACRO to include a positive response to the SAVESYS parameter, or
- Issuing the IPL command with the "SAVESYS systemname" parameter.

The "systemname" is the name assigned to the saved system. This is the same system name specified in the DMKSNT file.

The CMS S-disk must be mounted and attached to the virtual machine creating the saved system before the SAVESYS function is invoked. This ensures that CMS file directories are saved correctly. However, in addition to the S-disk, you need the Y-disk to save the shared Y-disk directory.

Note: Any updates to the CMS S-disk or Y-disk require saving the CMS system again.

In VM/SP, the CMS system is designed to be used as a saved system. Its location may be modified by an installation for its particular requirements but should be shared among CMS users.

Saved System Restrictions for CMS

Several coding restrictions must be imposed on CMS if it is to run as a saved system.

If the key specified in the CAW for a SIO instruction is zero, the data area for input may not cross the boundary between two pages with different storage keys.

If you intend to modify a shared CMS system, be sure that all code that is shared resides in the shared segments of the CMS nucleus. You can use the USERSECT area of DMSNUC to contain nonshared instructions.

If you want to have a system profile EXEC, or if you want to use the NOSPROF, INSTSEG, or SAVESYS parameters of the IPL command, you must have:

```
PARMRGS=(0,15)
```

in the NAMESYS macro for the CMS named system.

Using the System Profile, SYSPROF EXEC, for Tailoring

What the System Profile Does

The system profile is an EXEC that performs some of the CMS initialization function previously done in a module. You, the system programmer, can tailor the CMS environment that users see at initialization time by modifying this EXEC. You can do such things as access additional system disks or bring up application programs automatically. Decisions on tailoring the environment can be made on the basis of userid, responses to prompting, CMS parameters on the IPL command, or any other conditions defined by your installation.

You can also bypass the system profile. The system profile is not meant to force users into an environment. Instead, it enables you to change the default CMS environment established by DMSINS without modifying a CMS module and rebuilding CMS. You can modify the EXEC to make it as secure as your installation requires.

The CMS initialization module, DMSINS, calls SYSPROF EXEC and executes it from a DCSS (discontiguous shared segment) or the system disk or the system disk extension. This is done before any user disks are accessed. When you enter the IPL CMS command, SYSPROF EXEC is executed unless you:

- Specify the NOSPROF parameter on the command line, or
- IPL a non-DASD virtual device, such as a reader, or

- Do not have SYSPROF EXEC in the CMS search order.

The default SYSPROF EXEC creates an environment similar to the one DMSINS creates.

System Profile Functions

The default SYSPROF EXEC performs the following functions:

- Processes the parameters passed on the IPL command
- Displays the CMS system identification (system ID)
- Issues the initial console read
- Handles the first command entered at this read
- Accesses the 191 disk as the A-disk
- Accesses the 192 disk as the D-disk
- Issues the S-STAT/Y-STAT messages
- Issues other initialization related messages
- Executes the PROFILE EXEC if the user has one

If a protected user drops into CP, CP automatically re-IPLs. Information indicating the nature of the problem is passed to SYSPROF EXEC. SYSPROF EXEC displays a message when one of these conditions is found. However, by modifying the EXEC, you may choose to take different action, such as sending a message to an administrator.

You can also modify the SYSPROF EXEC to tailor your system to the requirements of your installation. For additional information on tailoring your system, see the *VM/SP CMS User's Guide.*

Invoking the IPL CMS Command

You can put the IPL CMS command in your directory entry or issue it after you log on. The IPL command has a PARM keyword marking the start of any CMS parameters. These parameters may be up to 64 bytes of data (excluding all leading blank characters after the keyword, PARM, but including all other embedded and trailing blanks).

Note: If you are IPLing a non-DASD device, such as a reader, all CMS parameters are ignored.

All parameters following the PARM keyword, except the SAVESYS parameter, are passed to SYSPROF EXEC. When a valid SAVESYS parameter is encountered (refer to the IPL command section in the *VM/SP CP Command Reference*), an attempt is made to save the system, and the

SAVESYS parameter is cleared. When an invalid SAVESYS parameter is encountered, all parameters are ignored. However, they are available to SYSPROF EXEC. DMSINS ignores any unrecognized parameters, but you can modify SYSPROF EXEC to recognize them.

To properly use the system profile and the new CMS parameters, `PARMRGS=(0,15)` must be coded in the NAMESYS macro for the CMS named system.

The CMS Parameters on the IPL Command

The format of the CMS parameters on the IPL command is:

IPL	*vaddr* / *sysname* [**PARM** [**SAVESYS** *sysname*]]

where:

(For other CMS parameters and any CP parameters, refer to the *VM/SP CP Command Reference.*)

vaddr
: is the virtual address (cuu) of the device containing the nucleus to be loaded. If vaddr is the address of a non-DASD virtual device, such as a virtual reader, any parameters specified after the PARM operand are ignored.

sysname
: is the name of a system previously saved by the SAVESYS command or parameter.

PARM
: is a CP keyword. CP passes up to 64 bytes of data (excluding all leading blank characters after the keyword, PARM, but including all other embedded and trailing blanks) to your virtual machine's general registers (4 bytes per register). For IPL of a virtual device, the movement begins with the high order byte of general register 0. For a named system, the place and length of movement is determined by the PARMRGS option specified in the NAMESYS entry.

SAVESYS *sysname*
: is used in the process of creating named systems. It saves a virtual machine storage space with registers and PSW as they currently exist. ***sysname*** must be a predefined name representing a definition of installation requirements of the named system. The definition indicates the number of pages to be saved, the DASD volume on which the system is to be saved, and the shared segments, if any. You must have the CP privilege class (class E) to issue the CP SAVESYS command.

After the system is saved, initialization continues, all CMS parameters entered on the IPL command are ignored, and SYSPROF EXEC is executed. The parameters entered on the original IPL command are not passed to SYSPROF EXEC.

No other parameters may be specified with this parameter. If any other parameters are specified, all parameters are ignored and a CONFLICT parameter is passed to SYSPROF EXEC, as well as the parameters entered on the IPL command.

How to Save a Named System

You must have the CP privilege class (class E) to issue the SAVESYS command. To save a named system, you can either:

- provide a positive response to the following message (message 729R),

  ```
  Do you want to save the system? (enter 1=YES or 0=NO)
  ```

  or

- enter the SAVESYS parameter in the PARM field of the IPL command.

If you specify the SAVESYS parameter on the IPL command, you should not specify any other parameters. If any other parameters are specified, they are all ignored but they are available to SYSPROF EXEC, and the system won't be saved.

How to Create or Change SYSPROF EXEC

VM/SP provides a default SYSPROF EXEC that may reside in a DCSS (discontiguous shared segment) or on the system disk or system disk extension. If it resides on a system disk, it must have a filemode of 2. For better performance, SYSPROF EXEC should reside in a DCSS. However, if it resides in a DCSS and a user IPLs with INSTSEG NO, SYSPROF EXEC is not invoked unless a copy also resides on the system disk or system disk extension.

The system profile is shipped as a SYSPROF $EXEC file. SYSPROF $EXEC is a fixed format file with a logical record length of 80. To change SYSPROF EXEC, edit the SYSPROF $EXEC file using the System Product Editor (XEDIT) with the UPDATE option. After making changes, use the EXECUPDT command with the SID option to apply the updates to the $EXEC file and to generate the updated SYSPROF EXEC.

If the EXEC resides on a system disk, it can be changed by anyone who has write access to that disk. Once the EXEC has been modified, if the system is to be IPLed as a named system, or if the S-STAT or Y-STAT capability is to be available, the system must be resaved. If the EXEC resides in a DCSS, the administrator must follow documented procedures for rebuilding and saving the DCSS after making changes to the EXEC. In any case, all users currently logged on must re-IPL the system or unpredictable results occur. Therefore, it is recommended that any logic in SYSPROF EXEC that changes frequently should

- Be put in another EXEC that is called by the system profile, and
- Placed on a disk other than the S- or Y-disks.

Modifications to EXECs that reside on other disks do not necessitate the resaving of the system and re-IPL by all users. However, in SYSPROF EXEC, you must remember to access the disks these EXECs reside on.

The following table lists the parameters that may be passed to SYSPROF EXEC:

Parameter	Meaning
AUTOCR	AUTOCR parameter specified on IPL command
BATCH	BATCH parameter specified on IPL command
NOSSTAT	S-STATs are unavailable
NOYSTAT	Y-STATs are unavailable
ACCD	Go ahead and try to access the D-disk
FORM	D-disk has already been formatted by DMSINS
NOWM iucv-rc	IUCV failed, with return code iucv-rc, windowing can't be started
SAVERR	SAVESYS specified without system name
CONFLICT	SAVESYS specified with other parameters
INSEGER1	INSTSEG value is missing and no shared EXECs are loaded.
INSEGER2	The Installation DCSS could not be loaded.
X'FF'	Delimiter for CP restart information
CMSERROR	CP entered; Information unavailable
DWAITPSW psw	CP entered; Disabled wait PSW 'psw'
EXTINTLP psw	CP entered; External interrupt loop
PAGERROR	CP entered; Paging error
PRGINTLP psw	CP entered; Program interrupt loop
SPAGEALT sname hexloc	CP entered; sname shared page hexloc altered
TRANEXCP	CP entered; Translation exception: While in non-EC mode
NOTREIPL	This was not an automatic re-IPL
X'FF'	Delimiter for CMS system ID
32 character string	CMS system ID
X'FF'	Delimiter for CMS parameters on IPL command
IPL parms	Data entered in PARM field of IPL command

Figure 38. Parameters Passed to SYSPROF EXEC

Some of these parameters cause messages to be displayed. You can modify SYSPROF EXEC to take further action when one of these parameters is found or to ignore these parameters completely.

When CP re-IPLs, it passes restart information to SYSPROF EXEC. This information indicates the nature of the problem. The default system profile issues a warning message when it finds this information. However, you can modify the EXEC to take different action.

How to Bypass the System Profile

To bypass the system profile, you can specify the NOSPROF parameter in the PARM field of the IPL command or IPL a non-DASD device.

If no SYSPROF EXEC exists, DMSINS performs CMS initialization. A warning message is displayed to notify the user of this condition.

Setting Up a Protected Application Environment

For users interested in only using application programs, it is possible to build a protected application environment. A protected user is automatically placed in their application environment at logon time and prevented from inadvertently dropping into CP. To set up this type of environment, you (the system programmer) should use the CONCEAL entry on the OPTION control statement in the user's directory. This prevents entry into CP by errors.

You should tailor SYSPROF EXEC, or any of the EXECs that it calls, to set up the proper application environment for the user. This environment can be set up based on userids, parameters passed in the PARM field of the IPL command, or any other function. Tailoring can range from simply accessing additional disks to suppressing normal CMS initialization messages and completely tailoring an application environment.

You should decide what degree of protection is required to make this environment. For example, you may want to turn off the CMS immediate commands such as HX, HI, and TS, for the duration of SYSPROF EXEC. However, keep in mind that since this must be done within SYSPROF EXEC, there is a short period of time where the user may type in one of these immediate commands during CMS initialization.

If the protected user drops into CP, CP re-IPLs. Information indicating the nature of the problem is passed to SYSPROF EXEC. Based upon the user and the reason for entry into CP, you may want to take action. For example, you may want to inform the system administrator.

What the System Profile Can Do for Installations

The system profile can set up several environments on a single system. Here are some examples of functions that can be done with SYSPROF EXEC at initialization time:

- New parameters can be recognized in the PARM field of the IPL command. For example, you can add the following IPL statement in a user's directory:

  ```
  IPL CMS PARM PROFS
  ```

  Since this is an unrecognized parameter, it would be ignored by DMSINS and passed to SYSPROF EXEC. Then, this parameter can be recognized and PROFS can be set up for the particular user. By recognizing several parameters, different environments can be tailored.

- Additional system disks can be accessed, or some defaults currently provided can be changed.
- Specific users or groups of users can be recognized and placed into an application or other environment.
- The initial console read can be suppressed, or the default can be changed to AUTOCR.
- Novice users can be prompted for information.
- Certain system messages, such as the CMS system ID, can be suppressed or overridden to hide the complexity of the system.
- Installations can decide for themselves how to handle conditions where protected users enter CP and are re-IPLed. For example, a message can be sent to the system administrator.

For additional information on the system profile see the *VM/SP CP Command Reference* and *VM/SP Installation Guide.*

Sharing File Directory Information

The SAVEFD command allows you to place file directory information for a shared, extended data format (EDF) R/O minidisk into a discontiguous shared segment (DCSS). The DCSS is then available to users who access the disk R/O. Placing the minidisk directory information in shared storage helps to reduce your working set and the system paging rate and response time.

The SAVEFD Command

SAVEFD performs the following tasks:

- The label record of the minidisk is updated with the name of the DCSS containing the file information.
- The hyperblock map and complete set of file status tables (FSTs) for the minidisk are placed into the DCSS.
- An ADT image is placed into the DCSS. When you access the minidisk, the ADT image is copied into nonshared storage, thereby picking up the information it contained exactly as CMS originally built it. Several fields in the ADT are then updated as part of ACCESS processing.

The format of the SAVEFD command is:

```
SAVEFD   { INIT   vdev label segname }
         { SAVE   vdev label segname }
         { NOSAVE vdev label         }
```

where:

INIT
: initializes the disk for a subsequent SAVEFD command. It accesses a disk and writes the segment name on the disk label record.

SAVE
: saves the file directory information in a DCSS. It accesses a disk; builds the ADT, FST hyperblocks, hypermap, and a SFD header in a DCSS address range; and issues a SAVESYS for the segment.

NOSAVE
: disables the saved storage access. It accesses a disk and clears the segment name on the disk label record.

vdev
: is the address of the disk for which saved file status table blocks (FSTBs) are to be built (the disk must be linked R/W).

label
: is a CMS minidisk label.

segname
: is the name of the shared segment.

Putting File Directory Information into Shared Storage

To put file directory information into shared storage, follow these steps:

1. Define the segment name in DMKSNT, and rebuild CP. For details on defining segment names in DMKSNT, see the *VM/SP Planning Guide and Reference.* For details on rebuilding CP, see the *VM/SP Installation Guide.*

2. Issue 'SAVEFD INIT' to write the segment name on the disk label record.

3. Issue 'SAVEFD SAVE' to save the FSTs in the segment.

   Note: Every time you update a SAVEFD minidisk, you must resave it by issuing 'SAVEFD SAVE'. You do not have to issue 'SAVEFD INIT' if there is no change in the segment name.

Usage Notes

1. Conventional data format (CDF) disks are not supported. If you use SAVEFD to process a CDF disk, the command terminates with a diagnostic message (DMSSFD260E).

2. To decide how big the segment should be for a SAVEFD minidisk, the following considerations should be made:

   - Each segment size is a multiple of 64K.

   - Each FST consists of 64 bytes; therefore, the number of FSTs in the disk is multiplied by 64.

   - Extra storage is needed for the SFD header, ADT, and hypermap.

   Each 64K segment may contain approximately 1000 files of file directory information.

3. To issue 'SAVEFD SAVE', you must be authorized to perform the SAVESYS operation.

4. When you make a change to a SAVEFD minidisk, CMS automatically updates the FST directory date/time stamp. Therefore, the date/time stamp of the FST directory of the SAVEFD minidisk and the date/time stamp in the SFD header in the DCSS are different. If other users access this SAVEFD minidisk now, they have nonsaved access. You should not access the disk as R/W unless you are updating and saving the directory information in the DCSS. After the changes are made, reissue the SAVEFD command to allow saved access again. The SAVEFD command accesses the disk as R/W.

5. The size of your virtual storage cannot be greater than the address of the DCSS where the FSTs are saved.

6. The SAVEFD command accesses the SAVEFD minidisk as a Z-disk. Therefore, any disk accessed as a Z-disk is released when SAVEFD is issued.

7. Auxiliary directories are not supported on disks for a saved storage access. You should not use SAVEFD on disks that use auxiliary directories. For details, see "Chapter 13. Using Auxiliary Directories" on page 339.

Messages and Return Codes

DMSSFD014E	Invalid function *function* [RC=24]
DMSSFD017E	Invalid device address *vdev* [RC=24]
DMSSFD047E	No function specified [RC=24]
DMSSFD050E	Parameter missing after *value* [RC=24]
DMSSFD070E	Invalid parameter *parameter* [RC=24]
DMSSFD109S	Insufficient free storage available [RC=41]
DMSSFD126S	Error {reading|writing} label on disk *mode(vdev)* [RC=100]
DMSSFD260E	Disk not properly formatted for SAVEFD [RC=16]
DMSSFD283E	The *name* DCSS could not be {found|loaded|saved}; return code *rc* from {FINDSYS|LOADSYS|SAVESYS} [RC=128]
DMSSFD286E	The DCSS is too small for the data being stored [RC=40]
DMSSFD288I	*dcssname* DCSS not saved
DMSSFD401S	VM size cannot exceed segment start address (*hex address*) [RC=40]
DMSSFD653E	Error executing *command*, rc=*nn* [RC=40]
DMSSFD1074S	Disk not linked as R/W [RC=36]
DMSSFD1075E	Label on disk *label* and label on command *label* do not match [RC=24]
DMSSFD1076E	Segment name in disk label *segname* and segment name on command *segname* do not match [RC=24]
DMSSFD1077E	Disk has not been initialized by SAVEFD INIT [RC=40]

Sharing EXECs and Editor Macros

You can place frequently used EXECs and Editor macros into a discontiguous shared segment (DCSS). This allows multiple users to share the same executing copy of the EXECs and Editor macros. The DCSSGEN command builds, loads, and saves the DCSS.

For more information, see the procedure for installing the CMSINST segment in the *VM/SP Installation Guide.*

Chapter 12. Using the CMS Batch Facility

The CMS Batch Facility is a VM/SP programming facility that runs under the CMS subsystem. It allows VM/SP users to run their jobs in batch mode by sending jobs either from their virtual machines or through the real (system) card reader to a virtual machine dedicated to running batch jobs. The CMS Batch Facility then executes these jobs freeing user machines for other uses.

If both the CMS Batch Facility and the Remote Spooling Communications Subsystem Networking Version 2 (RSCS) are being executed under the same VM/SP system, job input streams can be transmitted to the batch facility from remote stations via communication lines. Also, the output of the batch processing can be transmitted back to the remote station.

The CMS Batch Facility virtual machine is generated and controlled on a userid dedicated to execution of jobs in batch mode. The system operator generates the "batch machine" either by:

1. Entering the BATCH parameter in the PARM field of the IPL command, or

2. Specifying the NOSPROF parameter and entering CMSBATCH during VM READ.

The CMSBATCH module loads the DMSBTP TEXT S2 file, which is the actual batch processor. After each job is executed, the batch facility IPLs itself, thereby providing a continuously processing batch machine. The batch processor IPLs itself by using the PARM option of the CP IPL command followed by a character string that CMS recognizes as peculiar to a batch virtual machine performing its IPL. Jobs are sent to the batch machine's virtual card reader from users' terminals and executed sequentially. When there are no jobs waiting for execution, the CMS Batch Facility remains in a wait state ready to execute a user job. See the *VM/SP Operator's Guide* for more information about controlling the batch machine.

The CMS Batch Facility is particularly useful for compute-bound jobs such as assemblies and compilations and for execution of large user programs, since interactive users can continue working at their terminals while their time-consuming jobs are run in another virtual machine.

The system programmer controls the batch facility virtual machine environment by resetting the CMS Batch Facility machine's system limits, by writing routines that handle special installation input to the batch facility, and by writing EXEC procedures that make the CMS Batch Facility easier to use.

Installing the CMS Batch Machine

Before using the CMS Batch Facility, an entry must exist in the user's directory. This entry specifies the userid of the CMS batch machine.

Following is an example of a user directory entry granting authorization to use the CMS Batch Facility. Consult the *VM/SP Planning Guide and Reference* for the proper coding of the directory macro parameters.

```
USER CMSBATCH BATCH 1M 2M BG
     ACCOUNT 13 SYSTEM
     OPTION ACCT
     IPL CMS
     CONSOLE 009 3215
     SPOOL 00C 2540 READER *
     SPOOL 00D 2540 PUNCH A
     SPOOL 00E 1403 A
     LINK MAINT 190 190 RR
     MDISK 195 3330 xxx 010 'paswrd' W 'rdpswd' 'wrtpswd' 'allpswd'
```

In the example above, CMSBATCH is the userid of the CMS Batch machine and B and G are the privilege classes assigned to it. Privilege class B gives the CMS Batch machine MSGNOH capability. MSGNOH (Message No Header) strips the timestamp and userid from the messages you send. For information on redefining privilege classes, see the *VM/SP CP for System Programming.*

Note: There is no 191 MDISK for the CMS batch machine.

To have the CMS batch machine automatically logged on, you should have the following entry in the autolog virtual machine's (AUTOLOG1) PROFILE EXEC:

```
AUTOLOG CMSBATCH BATCH CMSBATCH
```

Otherwise, the operator logs on to the CMS batch machine and enters "CMSBATCH" followed by "DISCONNECT" (if the CMS batch machine is to run in DISCONNECT status). Refer to the *VM/SP CP Command Reference* for more information on the AUTOLOG command.

Resetting the CMS Batch Facility System Limits

Each job running under the CMS Batch Facility is limited by default to the maximum value of 32,767 seconds of virtual processor time, 32,767 punched cards output, and 32,767 printed lines of output. You can reset these limits by modifying the BATLIMIT MACRO file, which is found in the CMSLIB macro library, and by reassembling DMSBTP.

Writing Routines To Handle Special Installation Input

The CMS Batch Facility can handle user-specified control language and special installation batch facility /JOB control cards. These handling mechanisms are built into the system in the form of user exits from batch. You are responsible for generating two routines to make use of them. These routines must be named BATEXIT1 and BATEXIT2, respectively, and must have a filetype of TEXT and a filemode number of 2 if placed on the system disk or an extension of the system disk. (See the *VM/SP CMS User's Guide* for information on how to write and use CMS Batch Facility control cards.) The routines you write are responsible for saving registers, including general purpose register 12, which saves addressability for the batch facility. These routines (if made available on the system disk) are included with the CMS Batch Facility each time it is loaded.

BATEXIT1: Processing User-Specified Control Language

BATEXIT1 is an entry point provided so that users may write their own routine to check non-CMS control statements. For example, a routine could be written to scan for the OS job control language needed to compile, link edit, and execute a FORTRAN job. BATEXIT1 receives control after each read from the CMS Batch Facility virtual card reader is issued. General purpose register 1 contains the address of the batch facility read buffer, which contains the card image to be executed by the batch facility. This enables BATEXIT1 to scan each card it receives as input for the type of control information you specify.

If, after the card is processed by BATEXIT1, general purpose register 15 contains a nonzero return code, the CMS Batch Facility flushes the card and reads the next card. If a zero is returned in general purpose register 15, the batch facility continues processing by passing the card to CMS for execution.

BATEXIT2: Processing the Batch Facility /JOB Control Card

BATEXIT2 is an entry point provided so that users can code their own routine to use the /JOB card for additional information. BATEXIT2 receives control before the VM/SP routine used to process the batch facility /JOB card begins its processing, and BATEXIT2 receives control after CMS has scanned the /JOB card and built the parameter list. When BATEXIT2 is processing, general purpose register 1 points to the CMS parameter list buffer. This buffer is a series of 8-byte entries, one for each item on the /JOB card. If the return code found in general purpose register 15 resulting from BATEXIT2 processing of this card is nonzero, an error message is generated and the job is flushed. If general purpose register 15 contains a zero, normal checking is done for a valid userid and the existence of an account number. Finally, execution of this job begins.

EXEC Procedures for the Batch Facility Virtual Machine

You can control the CMS Batch Facility virtual machine using EXEC procedures. For example, you can use an EXEC:

- To produce the proper sequence of CP/CMS commands for users who do not know CMS commands and controls.
- To provide the sequence of commands needed to execute the most common jobs (assemblies and compilations) in a particular installation.

For information on how to use the EXEC facility to control the batch facility virtual machine, see the *VM/SP CMS User's Guide.*

Data Security under the Batch Facility

After each job, the CMS Batch Facility loads (via IPL) itself destroying all nucleus data and work areas. All disks where links were established during the previous job are detached.

At the beginning of each job, the batch facility work disk is accessed and then immediately erased preventing the current user job from accessing files that might remain from the previous job. Because of this, execution of the PROFILE EXEC is disabled for the CMS Batch Facility machine. You may, however, create an EXEC procedure called BATPROF EXEC and store it on any system disk to be used instead of the ordinary PROFILE EXEC. The batch facility then executes this EXEC at each job initialization time.

Improved IPL Performance Using a Saved System

Since the CMS batch processor goes through an IPL procedure after each user job, an installation may experience a more efficient IPL procedure by using a saved CMS system when processing batch jobs.

This can be accomplished by passing the name of the saved system to the CMS Batch Facility via the optional "sysname" operand in the CMSBATCH command line.

The batch facility saves the name of the saved system until the end of the first job. At this time it stores the name in the IPL command line both as the "device address" and as the PARM character string. The latter entry informs the CMS initialization routine (DMSINS) that a saved system has been loaded and that the name is to be saved for subsequent IPL procedures.

Note: When using the CMS SET command, the BLIP operand is ignored when issued from the CMS batch machine.

Chapter 13. Using Auxiliary Directories

When a disk is accessed, each module that fits the description specified on the ACCESS command is included in the resident directory. An auxiliary directory is an extension of the resident directory and contains the name and location of certain CMS modules that are not included in the resident directory. These modules, if added to the resident directory, would significantly increase its size, thus increasing the search time and storage requirements. An auxiliary directory can reference modules that reside on the S-disk; or, if the proper linkage is provided, reference modules that reside on any other read-only CMS disk. To take advantage of the saving in search time and storage, modules that are referenced via an auxiliary directory should never be in the resident directory. The disk where these modules reside should be accessed in a way that excludes these modules.

Adding an Auxiliary Directory

To add an auxiliary directory to CMS, the system programmer must generate the directory, initialize it, and establish the proper linkage. Only when all three tasks are completed, can a module described in an auxiliary directory be properly located.

Generating the Auxiliary Directory

An auxiliary directory TEXT deck is generated by assembling a set of DMSFST macros, one for each module name.

The DMSFST Macro

The format of the DMSFST macro is:

[*label*]	**DMSFST**	{ (*filename* [*,filetype* / <u>**,MODULE**</u>]) } [*,aliasname*] [**,FORM=E**]

where:

filename,filetype
: is the name of the module whose file status table (FST) information is to be copied.

aliasname
: is another name for the module.

FORM=E
: specifies that 64-byte FST entries are to be generated rather than 40-byte entries. Either length FST entry operates correctly on basic CMS. However, the 40-byte form does not contain such information as date/time after initialization by GENDIRT.

Initializing the Auxiliary Directory

After the auxiliary directory is generated via the DMSFST macro, it must be initialized. The CMS GENDIRT command initializes the auxiliary directory with the name and location of the modules to reside in an auxiliary directory. By using the GENDIRT command, the file entries for a given module are loaded only when the module is invoked.

Note: Do not load the modules into the transient area before issuing the GENDIRT command.

The GENDIRT Command

The format of the GENDIRT command is:

GENDIRT	*directoryname* [*targetmode* [*sourcemode*]]

where:

directoryname
: is the entry point of the auxiliary directory.

targetmode
: is the mode of the disk containing the modules referenced in the auxiliary directory. The letter is the mode of the disk containing the modules at execution time, not the mode of the disk at the initialization of the directory. At directory creation, all modules named in the directory being generated must be on either the A-disk on a read-only extension or on the disk specified in the sourcemode parameter. The default value for targetmode is S, the system disk. It is your responsibility to determine the usefulness of this operand at your installation and to inform users of programs using auxiliary directories of the proper method(s) of access.

sourcemode
: is the mode of the disk containing the modules or files when the GENDIRT command is issued. If not specified, 'A' is the default.

Establishing the Proper Linkage

The CMS module, DMSLAD, entry point DMSLADAD, must be called by a user program or interface to initialize the directory search order. The subroutine, DMSLADAD, must be called via an SVC 202 with register 1 pointing to the appropriate PLIST. The disk containing the modules listed in the auxiliary directory must be accessed as the mode specified, or implied, by the GENDIRT command before the call is issued. If the GENDIRT command has not been used, the user receives the message: "File not found" or "Error reading file".

The coding necessary for the call is:

```
       LA    R1,PLIST
       SVC   202
       DC    AL4(error return)
```

This call must be executed before the call to any module to be located via an auxiliary directory.

The PLIST should be:

```
PLIST  DS    0F
       DC    CL8'DMSLADAD'
       DC    V(directoryname)
       DC    F'0'
```

The auxiliary directory is copied into nucleus free storage. The active disk table (ADT) for the targetmode expressed or implied by the GENDIRT command is found and its file directory address chain (ADTFDA) is modified to include the nucleus copy of the auxiliary directory. A flag, ADTPSTM, in ADTFLG2 is set to indicate that the directory chain has been modified.

The address of the nucleus copy of the auxiliary directory is saved in the third parameter of the input PLIST and the high-order byte of the third parameter is set to X'80' to indicate that the directory search chain was modified and that the next call to DMSLADAD is a clear request.

To reset the directory search chain, a second call is made to DMSLADAD using the modified PLIST. DMSLADAD removes the nucleus copy of the auxiliary directory from the chain and frees it. This call to DMSLADAD removes all auxiliary blocks from the directory chain; there is no linkage to delete selective auxiliary directory blocks from the chain. DMSLADAD does not, however, restore the caller's PLIST to its initial state.

Error Handling and Return Codes

An error handling routine should be coded to handle nonzero return codes from DMSLADAD in register 15. The following errors (with condition code = 2) may occur on a call to DMSLADAD:

Description	DMSLADAD Request Type	Return Code
The auxiliary directory address is not specified in the PLIST.	initialize, clear	1
The targetmode specified at GENDIRT time is not accessed.	initialize	1
The targetmode specified at GENDIRT time is used to access an OS or DOS disk.	initialize	1
The address of the nucleus copy of the auxiliary directory is not specified in the third parameter of the PLIST.	clear	2
No auxiliary directory has been initialized on the given disk.	clear	2
The targetmode specified at GENDIRT time is accessed in shared storage.	initialize	3

Creating an Auxiliary Directory

In this example, consider an application called PAYROLL consisting of several modules. It is possible to put these modules in an auxiliary directory rather than in the resident directory. It is further possible to put the auxiliary directory on a disk other than the system disk. In this example, the auxiliary directory is placed on the Y-disk.

First, generate the auxiliary directory TEXT deck for the payroll application using the DMSFST macro:

```
PAYDIRT  START   0
         DC      F'40'  LENGTH OF FST ENTRY¹
         DC      A(DIRTEND-DIRTBEG) SIZE OF DIRECTORY
DIRTBEG  EQU     *
         DMSFST  PAYROLL1
         DMSFST  PAYROLL2
         DMSFST  PAYROLL3
         DMSFST  PAYFICA
         DMSFST  PAYFEDTX
         DMSFST  PAYSTATE
         DMSFST  PAYCITY
         DMSFST  PAYCREDU
         DMSFST  PAYOVERT
         DMSFST  PAYSICK
         DMSFST  PAYSHIFT
         DC      2A(0) POINTER TO NEXT FST BLOCK
DIRTEND  EQU     *
         END
```

[1]Note: F'64' should be used if FORM=E is specified on DMSFST macro.

In this example, the payroll control program (PAYROLL), the payroll auxiliary directory (PAYDIRT), and all the payroll modules reside on the 194 disk.

In the payroll control module (PAYROLL), the subroutine DMSLADAD must be called to establish the linkage to the auxiliary directory. This call must be executed before any call is made to a payroll module that is in the PAYDIRT auxiliary directory.

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

PLIST     DS    0F
          DC    CL8'DMSLADAD'
          DC    V(PAYDIRT)
          DC    F'0'
```

Next, all payroll modules must have their absolute core-image files generated and the payroll auxiliary directory must be initialized. In the example, the payroll control module (PAYROLL) is given a mode number of 2 while the other payroll modules are given a mode number of 1. When the PAYROLL program is finally executed, only the files on the 194 disk with a mode number of 2 are accessed. This means only the PAYROLL control program (which includes the payroll auxiliary directory) will be referenced from the resident directory. All the other payroll modules, because they have mode numbers of 1, are referenced via the payroll auxiliary directory.

The following sequence of commands create the absolute core-image files for the payroll modules and initialize the payroll auxiliary directory.

```
ACCESS 194 A
LOAD PAYROLL PAYDIRT
GENMOD PAYROLL     (now the auxiliary directory is included
                   in the payroll control module, but it is
                   not yet initialized.)
LOADMOD PAYROLL
INCLUDE PAYROLL1
GENMOD PAYROLL1    (this sequence of three commands is
   .               repeated for each payroll module called
   .               by PAYROLL to establish the proper
   .               address where the module would be loaded.)
LOADMOD PAYROLL
INCLUDE PAYSHIFT
GENMOD PAYSHIFT

LOADMOD PAYROLL
GENDIRT PAYDIRT Y
GENMOD PAYROLL MODULE A2
```

When it is time to execute the PAYROLL program, the 194 disk must be accessed as the Y-disk (the same mode letter as specified on the GENDIRT command). Also, the 194 disk is accessed in a way that includes the PAYROLL control program in the resident directory but not the other payroll modules. This is done by specifying a mode number of 2 on the ACCESS command.

```
ACCESS 194 Y/S * * Y2
```

Now, a request for a payroll module, such as PAYOVERT, can be successfully fulfilled. The auxiliary directory will be searched and PAYOVERT will be found on the Y-disk.

Notes:

1. *A disk referred to by an auxiliary directory must be accessed as a read-only disk, and it cannot be accessed in shared storage. (For details on the ACCESS command, see the VM/SP CMS Command Reference. For details on the SAVEFD command, see "Sharing File Directory Information" on page 328.)*

2. *You cannot issue the GENDIRT command against an auxiliary directory included in a transient module, since the GENDIRT command is also a transient module. In the example above, if you issue:*

   ```
   ACCESS 194 A
   LOAD PAYROLL PAYDIRT (ORIGIN TRANS
   .
   .
   .
   LOADMOD PAYROLL
   GENDIRT PAYDIRT Y
   ```

   the GENDIRT module overlays the PAYROLL module in the transient area before initializing the PAYDIRT auxiliary directory, and hence, would fail.

Chapter 14. Understanding Assembler Virtual Storage Requirements

The minimum size virtual machine required by the assembler is 256K bytes. However, better performance is generally achieved if the assembler is run in 320K bytes of virtual storage. This size is recommended for medium and large assemblies.

If more virtual storage is allocated to the assembler, the size of buffers and work space can be increased. The amount of storage allocated to buffers and work space determines assembler speed and capacity. Generally, as more storage is allocated to work space, larger and more complex macro definitions can be handled.

You can control the buffer sizes for the assembler utility data sets (SYSUT1, SYSUT2, and SYSUT3), and the size of the work space used during macro processing, by specifying the BUFSIZE assembler option. Of the storage given, the assembler first allocates storage for the ASSEMBLE and CMSLIB buffers according to the specifications in the DD statements supplied by the FILEDEF for the data sets. Then the assembler allocates storage for the modules of the assembler. The remainder of the virtual machine is allocated to utility data set buffers and macro generation dictionaries according to the BUFSIZE option specified:

BUFSIZE(STD):
: 37 percent is allocated to buffers, and 63 percent to work space. This is the default if you do not specify any BUFSIZE option.

BUFSIZE(MIN):
: Each utility data set is allocated a single 790-byte buffer. The remaining storage is allocated to work space. This allows relatively complex macro definitions to be processed in a given virtual machine size, but the speed of the assembly is substantially reduced.

Overlay Structures

An overlay structure can be created in CMS in two different ways, although CMS has no overlay supervision. For descriptions of all the CMS commands mentioned, see the *VM/SP CMS Command Reference.*

Prestructured Overlay

A prestructured overlay program is created using the LOAD, INCLUDE, and GENMOD commands. Each overlay phase or segment is a nonrelocatable core-image module created by GENMOD. The phases may be brought into storage with the LOADMOD command.

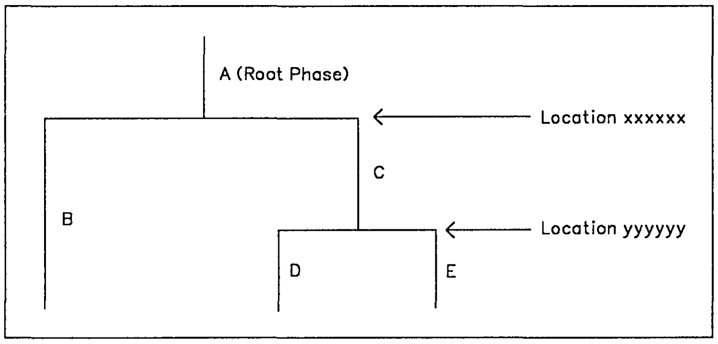


Figure 39. An Overlay Structure

The overlay structure shown in Figure 39 could be prestructured using the following sequence of commands (Programs A, B, C, D, and E are the names of TEXT files; the overlay phases will be named Root, Second, Third, etc.):

```
LOAD     A B
GENMOD   ROOT   (FROM A TO B STR)
GENMOD   SECOND (FROM B)
LOADMOD  ROOT
INCLUDE  C D
GENMOD   THIRD  (FROM C TO D)
GENMOD   FOURTH (FROM D)
LOADMOD  THIRD
INCLUDE  E
GENMOD   FIFTH  (FROM E)
```

The programmer need not know the storage address where each phase begins. A TEXT file can be made to load at the proper address by reloading earlier phases. In the foregoing example, the command sequences, "LOADMOD ROOT/INCLUDE C D" and "LOADMOD THIRD/INCLUDE E," cause TEXT files C, D, and E to load at the proper addresses.

If the root phase contains address constants to the other phases, one copy of the root must be kept in storage while each of the other phases is brought in by the LOAD or INCLUDE commands without an intervening GENMOD. The root phase is then processed by GENMOD after all address constants have been satisfied. In this case, the programmer must know the address where non-root phases begin (in Figure 39 on page 346, locations xxxxxx and yyyyyy). The following sequence of commands could be used:

```
LOAD     A B
GENMOD   SECOND (FROM B)
INCLUDE  C D    (ORIGIN, xxxxxx)
GENMOD   THIRD  (FROM C TO D)
GENMOD   FOURTH (FROM D)
INCLUDE  E      (ORIGIN yyyyyy)
GENMOD   FIFTH  (FROM E)
LOAD     A B
INCLUDE  C D    (ORIGIN xxxxxx)
INCLUDE  E      (ORIGIN yyyyyy)
GENMOD   ROOT   (FROM A TO C STR)
```

The ORIGIN option of the INCLUDE command is used to cause the included file to overlay a previously loaded file. The address at which a phase begins must be a doubleword boundary. For example, if the root phase were X'2BD' bytes long, starting at virtual storage location X'20000', then location xxxxxx would be the next doubleword boundary, or X'202C0'.

The STR option, which is specified in the GENMOD of the root phase, specifies that whenever that module is brought into storage with the LOADMOD command, the Storage Initialization routine should be invoked. This routine initializes user free storage pointers.

At execution time of the prestructured overlay program, each phase is brought into storage with the LOADMOD command. The phases can call LOADMOD. The OS macros LINK, LOAD, and XCTL normally invoke the INCLUDE command, which loads TEXT files. These macros will invoke LOADMOD if a switch, called COMPSWT, in the CMS nucleus constant area, NUCON, is turned on.

With COMPSWT set, overlay phases that use LINK, LOAD, and XCTL must be prestructured MODULE files.

Dynamic Load Overlay

The dynamic load method of using an overlay structure is to have all the phases in the form of relocatable object code in TEXT files or members of a TEXT library, filetype TXTLIB. The OS macros, LINK, LOAD, and XCTL may then be used to pass control from one phase to another. The XCTL macro causes the calling program to be overlayed by the called program except when it is issued from the root phase. When issued from the root phase, CMS treats XCTL as it would a LINK macro, adding the new code at the end of the root phase.

The COMPSWT flag in OSSFLAGS must be off when the dynamic load method is used.

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 CMSSERV communications end.
APPLMSG	Accesses and displays messages from a message 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 Personal Computer Enhanced Connectivity Facilities 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 DISPW Macro" on page 93.
DMSABN	Abend the virtual machine. Refer to "Chapter 2. Processing Abends" on page 5.
DMSEXS	Execute an instruction without nucleus protection. Refer to "The DMSEXS Macro" on page 41.
DMSFREE	Gets free storage. Refer to "The DMSFREE Macro" on page 27.
DMSFRET	Releases free storage. Refer to "The DMSFRET Macro" on page 33.
DMSFST	Sets up a file status table for a given file. Refer to "The DMSFST Macro" on page 339.
DMSKEY	Sets nucleus protection on or off. Refer to "The DMSKEY Macro" on page 39.
EPLIST	DSECT to map extended PLIST passed in register 0.

CMS Macro	Function
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.*
PUNCHC	Punches a card. Refer to the *VM/SP CMS Macros and Functions Reference.*

CMS Macro	Function
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.
TVSPARMS	Sets tape volume switching parameters for DMSTVS. Refer to "OS Tape Volume Switching" on page 204.
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. Sample Terminal Session for OS Programmers

The following terminal session shows how you might create an assembler language program in CMS, assemble it, correct assembler errors and execute it. All the lines in blue are lines that you should enter at the terminal. The other lines represent the system responses that you should receive when you enter the command.

The input data lines in the example are aligned in the proper columns for the assembler; if you are using a typewriter terminal, you should set your terminal's tab stops at columns 10, 16, 31, 36, 41, and 46, and use the Tab key when you want to enter text in these columns. If you are using a display terminal, when you use a PF key defined as a tab or some input character, the line image is expanded as it is placed in the screen output area.

There are some errors in the terminal session, so that you can see how to correct errors in CMS.

```
1   xedit ostest assemble
    Creating new file:
    input
    Input mode:
    dataproc csect
             print nogen
             space
    r0       equ   0
    r1       equ   1
    r2       equ   2
    r10      equ   10
    r12      equ   12
    r13      equ   13
    r14      equ   14
    r15      equ   15
```

----

1 The XEDIT command is issued to create a file named OSTEST ASSEMBLE. Since the file does not exist, the editor indicates that it is a new file, and you can use the INPUT subcommand to enter input mode and begin entering the input lines.

```
         space
         stm   r14,r12,12(r13)  save caller's regs
         balr  r12,0            establish
         using *,r12            addressability
         st    r13,savearea+4 store addr of caller's savearea
         la    r15,savearea   get the address of my savearea
         st    r15,8(r13)     store addr in caller's savearea
         lr    r13,r15        save addr of my savearea
         space
*open files and check that they opened okay
         space
         la    r3,0           initially set return code
         open  (indata,outdata,(output))     open files
         using ihadcb,r10     get dsect to check files
         la    r10,indata     prepare to check output file
         tm    dcboflgs,x'10' everything ok?
         bnz   checkout       ...continue
         la    r3,100         set return code
         b     exit           ...exit
checkout la    r10,outdata    check output file
         tm    dcboflgs,x'10' is it okay?
         bnz   process        ...
         la    r3,200         set return code
         b     exit
         space
process  equ   *
         get   indata         read a record from input file
         lr    r2,r1          save address of record
         put   outdata,(2)    move it to output
         b     process        continue until end-of-file
         space
exit     equ   *
         close (indata,,outdata)   close files
         l     r13,savearea+4 addr of caller's save area
         lr    r15,r3         load return code
         l     r14,12(r13)    get return address
         lm    r0,r12,20(r13) restore regs
         br    r14            bye...
         space
savearea dc    18f'0'
indata   dcb   ddname=indd,macrf=gl,dsorg=ps,recfm=f,lrecl=80,      *
```

```
2   save#input
    Input mode:
                        eodad=exit
    outdata  dcb   ddname=outdd,macrf=pm,dsorg=ps
             dcbd
             space
             end
3
    file
    Ready;
4   global maclib osmacro
    Ready;
5   assemble ostest
6   ASSEMBLER DONE
    OST00230       23         LA    R3,0          INITIALLY SET RETURN CODE
    IFO188 R3 IS AN UNDEFINED SYMBOL
    OST00240       24         OPEN  (INDATA,OUTDATA,(OUTPUT))    OPEN FILES
    4000000       27+   12,***  IHB002  INVALID OPTION OPERAND SPECIFIED-OUTDATA
    IFO197 *** MNOTE ***
    OST00290       32         LA    R3,100        SET RETURN CODE
    IFO188 R3 IS AN UNDEFINED SYMBOL
    OST00340       37         LA    R3,200        SET RETURN CODE
    IFO188 R3 IS AN UNDEFINED SYMBOL
    OST00460       63         LR    R15,R3        LOAD RETURN CODE
    IFO188 R3 IS AN UNDEFINED SYMBOL
    NUMBER OF STATEMENTS FLAGGED IN THIS ASSEMBLY =      5
    Ready(00012);
```

----

2 Before continuing to enter input lines, the EDIT subcommand SAVE is issued to write what has already been written onto disk. The CP logical line end symbol (#) separates the SAVE and INPUT subcommands.

3 A null line returns you to edit mode. You may wish, at this point, to proofread your input file before issuing the FILE subcommand to write the ASSEMBLE file onto disk.

4 Since this assembler program uses OS macros, you must issue the GLOBAL command to identify the CMS macro library, OSMACRO MACLIB, before you can invoke the assembler.

5 The ASSEMBLE command invokes the VM/SP assembler to assemble the source file.

6 The assembler displays errors encountered during assembly. Depending on how accurately you copied the program in this sample session, you may or may not receive some of these messages; you may also have received additional messages.

```
7    xedit ostest assemble
     locate /r2
     R2       EQU   2
     i r3       equ   3
     /open
     *OPEN FILES AND CHECK THAT THEY OPENED OKAY
     /open
              OPEN   (INDATA,OUTDATA,(OUTPUT))     OPEN FILES
     c /,/,,/
              OPEN   (INDATA,,OUTDATA,(OUTPUT))     OPEN FILES
8    file
     Ready;
     assemble ostest
9    ASSEMBLER DONE
     NO STATEMENTS FLAGGED IN THIS ASSEMBLY
     Ready;
10   filedef indd disk test data a
     Ready;
11   filedef outdd punch
     Ready;
12   cp spool punch to *
     Ready;
```

----

7 You must edit the file OSTEST ASSEMBLE and correct any errors in it. The errors placed in the example included a missing comma on the OPEN macro and the omission of an EQU statement for a general register. These changes are made as shown. The CMS Editor accepts a diagonal (/) as a LOCATE subcommand.

8 After all the changes have been made to the ASSEMBLE file, you can issue the FILE subcommand to replace the existing copy on disk and then reassemble it.

9 This time the assembler completes without encountering any errors. If your ASSEMBLE file still has errors, you should use the editor to correct them.

10 The FILEDEF command is used to define the input and output files used in this program. The ddnames INDD and OUTDD, defined in the DCBs in the program, must have a file definition in CMS. To execute this program, you should have a file on your A-disk named TEST DATA, which must have fixed-length, 80-character records. If you have no such file, you can make a copy of your ASSEMBLE file as follows:

```
copyfile ostest assemble a test data a
```

11 The output file is defined as a punch file so that it will be written to your virtual card punch.

12 The CP SPOOL command is issued, using the CP function, to spool your virtual punch to your virtual card reader.

```
13  load ostest
    Ready;
    start
    DMSLIO740I Execution begins...
14  DMSSOP036E Open error code 04 on OUTDD.
    Ready(00200);
15  filedef
    INDD     DISK     TEST     DATA     A1
    OUTDD    PUNCH
    Ready;
16  filedef outdd punch (lrecl 80 recfm f
    Ready;
17  cp query reader all
    NO RDR FILES
    Ready;
18  load ostest (start
    DMSLIO740I Execution begins...
19  PUN FILE 6198  TO  BILBO    COPY 01 NOHOLD
    Ready;
```

----

13 The LOAD command loads the TEXT file produced by the assembly into virtual storage. The START command begins program execution.

14 An open error is encountered during program execution. The CMS ready message indicates a return code of 200, which is the value placed in it by your program.

15 The FILEDEF command, with no operands, results in a display of the current file definitions in effect.

16 Error code 4 on an open request means that no RECFM or LRECL information is available. An examination of the program listing would reveal that the DCB for OUTDD does not contain any information about the file format; you must supply it on the FILEDEF command. Re-enter the FILEDEF command.

17 You can use the CP QUERY command to determine whether there are any files in your card reader. It should be empty; if not, determine whether they might be files you need and, if so, read them into your virtual machine; otherwise, purge them.

18 Use the LOAD command to execute the program again; this time, use the START option of the LOAD command to begin the program execution.

19 The PUN FILE message indicates that a file has been transferred to your virtual card reader. The ready message indicates that your program executed successfully.

```
20  fi indd reader
    Ready;
    fi outdd disk new osfile a4 (recfm fb block 1600 lrecl 80
    Ready;
21  listfile new osfile a4 (label
    DMSLST002E File not found.
    Ready(00028);
22  run ostest
    Execution begins...
    Ready;
23  listfile new osfile a4 (label
    FILENAME  FILETYPE  FM  FORMAT  LRECL  RECS BLOCKS   DATE     TIME     LABEL
    NEW       OSFILE    A4  F        1600     5     10  9/30/75  8:26:14  PAT198
    Ready;
```

----

20 For the next execution of this program, you are going to read the file back out of your card reader and create a new CMS disk file in OS simulated data set format. FI is an acceptable system truncation for the command named FILEDEF.

21 The LISTFILE command is issued to check that the file NEW OSFILE does not exist.

22 The RUN command (which is an EXEC procedure) is used instead of the LOAD and START commands to load and execute the program. The ready message indicates that the program completed execution.

23 The LISTFILE command is issued again, and the file NEW OSFILE is listed. (If you issue another CP QUERY READER command, you will also see that the file is no longer in your card reader.)

Appendix C. Sample Terminal Session for DOS Programmers

The following terminal session shows how you might create an assembler language program in CMS, assemble it, correct assembler errors and execute it. All the lines in blue are lines that you should enter at the terminal. The other lines represent the system responses that you should receive when you enter the command.

The input data lines in the example are aligned in the proper columns for the assembler; if you are using a typewriter terminal, you should set your terminal's tab stops at columns 10, 16, 31, 36, 41 and 46, and use the Tab key when you want to enter text in these columns. If you are using a display terminal, when you use a PF key or an input character defined as a tab, the line image is expanded as it is placed in the screen output area.

Note: The assembler, in CMS, cannot read macros from VSE/AF libraries. This sample terminal session shows how to copy macros from VSE/AF libraries and create CMS MACLIB files. Ordinarily, the macros you need should already be available in a system MACLIB file. You do not have to create a MACLIB each time you want to assemble a program.

There are some errors in the terminal session so that you can see how to correct errors in CMS.

```
1   cp link dosres 130 130 rr linkdos
    DASD 130 LINKED R/O
    Ready;
    access 130 z
    Z (130) R/O - DOS
    Ready;
2   set dos on z
    Ready;
```

----

1 Use the CP LINK command to link to the DOS system residence volume and the ACCESS command to access it. In this example, the system residence is at virtual address 130 and is accessed as the Z-disk.

2 Enter the CMS/DOS environment specifying the mode letter at which the DOS/VS (VSE/AF) system residence is accessed.

```
3    xedit dostest assemble
     Creating new file:
     input
     Input mode:
     begpgm   csect
              balr  12,0
              using *,12
              la    13,savearea
              open  infile,outfile
     loop     get   infile
              put   outfile
              b     loop
     eodad    equ   *
              close infile,outfile
              eoj
              eject
     buffer   dc    CL80' '
     infile   dtfdi modname=shrmod,ioarea1=buffer,devaddr=sysipt,       *
4    save#input
     Input mode:
                    eofaddr=eodad,recsize=80
     outfile  dtfdi modname=shrmod,ioarea1=buffer,devaddr=syspch,       *
     save#input
     Input mode:
                     recsize=81
     shrmod   dimod typefle=output
     endpgm   equ   *
              end
```

----

3 Use the EDIT command to create a file named DOSTEST ASSEMBLE. Since the file does not exist, the editor indicates that it is a new file and you can use the INPUT subcommand to enter input mode and begin entering the input lines.

4 Before continuing to enter input lines, the XEDIT subcommand SAVE is issued to write what has already been written onto disk. The logical line end symbol (#) separates the SAVE and INPUT subcommands. Another continuation character is needed.

```
5
    file
    Ready;
6   xedit getmacs eserv
    Creating new file:
    tabs 2 72
    input
    Input mode:
7   punch open,close,get,put,dimod,dtfdi

    file
    Ready;
8   assgn sysipt a
    Ready;
    eserv getmacs
    Ready;
9   listfile getmacs *
    GETMACS  ESERV    A1
    GETMACS  MACRO    A1
    GETMACS  LISTING  A1
    Ready;
10  maclib gen dosmac getmacs
    Ready;
    erase getmacs *
    Ready;
```

----

5 A null line returns you to edit mode. You may want, at this point, to proofread your input file before issuing the FILE subcommand to write the ASSEMBLE file on disk.

6 To obtain the macros you need to assemble this file, use the editor to create an ESERV file. By setting the logical tabs at columns 2 and 72, you can protect yourself from entering data in column 1.

7 PUNCH is an ESERV program control statement that copies and de-edits macros from source statement libraries; in this case, the system source statement library. The output is directed to the SYSPCH device, which the CMS/DOS ESERV EXEC assigns by default to your A-disk.

8 You must assign the logical unit SYSIPT before you invoke the ESERV command. GETMACS is the filename of the ESERV file containing the ESERV control statements.

9 After the ESERV EXEC completes execution, you have three files. You may want to examine the LISTING file to check the ESERV program listing. The MACRO file contains the punch (SYSPCH) output.

10 The MACLIB command creates a macro library named DOSMAC MACLIB. Since the MACLIB command completed successfully, you can erase the files GETMACS ESERV, GETMACS LISTING and GETMACS MACRO; an asterisk in the filetype field of the ERASE command indicates that all files with the filename of GETMACS should be erased.

```
11   global maclib dosmac
     Ready;
12   assemble dostest
13   ASSEMBLER DONE
     DOS00040        4          LA    13,SAVEAREA
     IFO188 SAVEAREA IS AN UNDEFINED SYMBOL
     DOS00110        35         EOJ
     IFO078 UNDEFINED OP CODE
     NUMBER OF STATEMENTS FLAGGED IN THIS ASSEMBLY =     2
     Ready(00008);
14   xedit dotest assemble
     locate /buffer/
     BUFFER    DC    CL80' '
     input savearea  ds    9d
     file
     Ready;
15   xedit eoj eserv
     Creating new file:
     i   punch eoj
     file
     Ready;
16   listio sysipt
      SYSIPT  DISK      A
     Ready;
     eserv eoj
     Ready;
```

----

11 Before you can invoke the assembler, you have to identify the macro library that contains the macros; use the GLOBAL command specifying DOSMAC MACLIB.

12 The ASSEMBLE command invokes the VM/SP assembler to assemble the source file.

13 The assembler displays errors encountered during assembly. Depending on how accurately you copied the program in this sample session, you may or may not receive some of these messages; you may also have received additional messages.

14 To correct the first error, which was the omission of a DS statement for SAVEAREA, edit the file DOSTEST ASSEMBLE and insert the missing line.

15 The second error indicates that the macro EOJ is not available since it was not copied from the source statement library. Create another ESERV file to punch this macro.

16 Use the LISTIO command to check that SYSIPT is still assigned to your A-disk so that you do not have to issue the ASSGN command again. Then issue the ESERV command again, this time specifying the filename EOJ.

```
17  maclib add dosmac eoj
    Ready;
    maclib map dosmac (term
    MACRO    INDEX  SIZE
    OPEN         2    43
    CLOSE       46    43
    GET         90    56
    PUT        147    93
    DIMOD      241   647
    DTFDI      889   284
    EOJ       1174     6
    Ready;
18  erase eoj *
    Ready;
    assemble dostest
19  ASSEMBLER DONE
    NO STATEMENTS FLAGGED IN THIS ASSEMBLY
    Ready;
20  listfile dostest *
    DOSTEST  ASSEMBLE A1
    DOSTEST  LISTING  A1
    DOSTEST  TEXT     A1
    Ready;
    print dostest listing
    Ready;
21  doslked dostest
    Ready;
```

----

17 Use the ADD function of the MACLIB command to add the macro EOJ to DOSMAC MACLIB. Then issue the MACLIB command again using the MAP function and the TERM option to display a list of the macros in the library.

18 Erase the EOJ files. You should always remember to erase files that you do not need any longer. Reassemble the program.

19 This time the assembler completes without encountering any errors. If you ASSEMBLE file still has errors, you should use the editor to correct them.

20 Use the LISTFILE command to check for DOSTEST files. The assembler created the files DOSTEST LISTING and DOSTEST TEXT. The TEXT file contains the object module. You can print the program listing if you want a printer copy. Then you may want to erase it.

21 Use the DOSLKED command to link-edit the TEXT file into an executable phase and write it into a DOSLIB. Since this program has no external references, you do not need to add any linkage editor control statements.

```
22  listfile dostest *
    DOSTEST  ASSEMBLE A1
    DOSTEST  DOSLIB   A1
    DOSTEST  TEXT     A1
    DOSTEST  LISTING  A1
    DOSTEST  MAP      A5
    Ready;
23  cp spool punch to *
    Ready;
    punch test data a
    PUN FILE 0100  TO BILBO    COPY 01 NOHOLD
    Ready;
    cp query reader all
    Ready;
    ORIGINID FILE CLASS RECDS  CPY HOLD DATE  TIME     NAME      TYPE    DIST
    PATTI    5840 A PUN 000097 01  NONE 09/29 15:00:39 TEST      DATA    BIN211
24  assgn sysipt reader
    Ready;
    assgn syspch a
    Ready;
25  dlbl outfile a cms punch output (syspch
    Ready;
    state punch output a
    DMSSTT002E File not found.
    Ready(00028);
```

----

22 Now you have a DOSTEST DOSLIB containing the link-edited phase and a MAP file containing the linkage editor map. You can display the linkage editor map with the TYPE command or use the PRINT command if you want a printer copy.

23 To execute this program in CMS/DOS, punch a file that has fixed- length, 80-character records into your virtual card punch. If you do not have any files that have fixed-length, 80-character records, you can create a file named TEST DATA with the CMS Editor or by copying your ASSEMBLE source file with the COPYFILE command as follows:

    copyfile dostest assemble a test data a

Use the CP SPOOL command to spool the punch to your own virtual machine, then use the PUNCH command to punch the file. The PUN FILE message indicates that the file is in your card reader. Use the CP QUERY command to check that it is the first or only file in your reader.

24 Use the ASSGN command to assign SYSIPT to your card reader and SYSPCH to your A-disk.

25 When you assign a logical unit to a disk mode, you must issue the DLBL command to identify the disk file to CMS. For this program execution, you are creating a CMS file named PUNCH OUTPUT. The STATE command ensures that the file does not already exist. If it does exist, rename it or else use another filename or filetype on the DLBL command.

```
26  global doslib dostest
    Ready;
    fetch dostest
    DMSFET710I Phase DOSTEST entry point at location 020000.
    Ready;
27  start
    DMSLIO740I Execution begins...
    Ready;
    listfile punch output a (label
    FILENAME  FILETYPE  FM  FORMAT  LRECL  RECS BLOCKS   DATE     TIME     LABEL
    PUNCH     OUTPUT    A1  F          80    97     10 9/29/79 14:50:55  BBB191
    Ready;
    cp query reader all
    Ready;
    NO RDR FILES
28  assgn sysipt a
    Ready;
    dlbl infile a cms punch output (sysipt
    Ready;
    assgn syspch punch
    Ready;
```

----

26 Use the GLOBAL command to identify the DOSLIB, DOSTEST if you want to search for executable phases; then issue the FETCH command specifying the phase name. The FETCH command loads the executable phase into storage. When the FETCH command is executed without the START option, a message is displayed indicating the entry point location of the program loaded.

27 The START command begins program execution. The CMS ready message indicates that your program completed successfully. You can check the input and output activity by using the LISTFILE command to list the file PUNCH OUTPUT. If you use the CP QUERY command, you can see that the file is no longer in your virtual card reader.

28 If you want to execute this program again, you can assign SYSIPT and SYSPCH to different devices; in this example, the input disk file PUNCH OUTPUT is written to the virtual punch. You do not need to reissue the GLOBAL DOSLIB command; it remains in effect until you reissue it or IPL CMS again.

```
29  fetch dostest (start
    DMSLIO740I Execution begins...
30  PUN FILE 5829  TO  BILBO    COPY 01 NOHOLD
    Ready;
    read punch2 output
    Ready;
    listfile punch2 output a (label
    FILENAME  FILETYPE  FM  FORMAT  LRECL   RECS BLOCKS   DATE     TIME     LABEL
    PUNCH2    OUTPUT    A1  F          80     97     10 9/29/75 14:50:59  BBB191
    Ready;
```

----

29 This time the program execution starts immediately because the START option is specified on the FETCH command.

30 Again, the PUN FILE message indicates that a file has been received in your virtual card reader. You can use the CMS command READCARD to read it onto disk and assign it a filename and filetype; in this example, PUNCH2 OUTPUT.

Appendix D. Sample Terminal Session Using Access Method Services

This sample terminal session shows you how to use access method services under CMS. You should have an understanding of VSAM and access method services before you use this terminal session.

The terminal session uses a number of CMS files, which you may create during the course of the terminal session; or, you may prefer to create all of the files that you need beforehand. Within the sample terminal session, the file that you should create is displayed prior to the commands that use it.

This terminal session is for both CMS OS VSAM programmers and CMS/DOS VSAM programmers. All entries in blue are entries you (a CMS OS VSAM programmer or a CMS/DOS VSAM programmer) should enter. The entries in blue and shaded are only for CMS/DOS programmers.

Notes:

1. *This terminal session assumes that you have, to begin with, a read/write CMS A-disk. This is the only disk required. Additional disks used in this exercise are temporary disks, formatted with the Device Support Facility program. If you have OS or DOS disks available, you should use them, and remember to supply the proper volume and virtual device address information, where appropriate. The number of cylinders available to users for temporary disk space varies among installations; if you cannot acquire ample disk space, see your system support personnel for assistance.*

2. *Output listings created by AMSERV take up disk space, so if your A-disk does not have a lot of space on it, you may want to erase the LISTING files created after each AMSERV step.*

3. *If any of the AMSERV commands that you execute during this sample terminal session issue a nonzero return code; for example:*

   Ready(00012);

   You should edit the LISTING file to examine the access method services error messages. The publication VSE/VSAM Messages and Codes contains the return codes and reason codes issued by access method services. You should determine the cause of the error, examine the DLBL commands and AMSERV files you used, correct any errors, and retry the command.

```
1   cp define t3330 200 10
    Ready;
    DASD 200 DEFINED
    cp query virtual 200
    Ready;
    DASD 200 3330 (TEMP) R/W     10 CYL

    cp define t3330 300 10
    Ready;
    DASD 300 DEFINED
    cp query virtual 300
    Ready;
    DASD 300 3330 (TEMP) R/W     10 CYL

    cp define t3330 400 10
    Ready;
    DASD 400 DEFINED
    cp query virtual 400
    Ready;
    DASD 400 3330 (TEMP) R/W     10 CYL

2   File: PUNCH DSF

     INIT UNIT(200) DEVTYP(3330) NVFY VOLID(222222) DVTOC(0,1,1) -
       MIMIC(MINI(10))
     INIT UNIT(300) DEVTYP(3330) NVFY VOLID(333333) DVTOC(0,1,1) -
       MIMIC(MINI(10))
     INIT UNIT(400) DEVTYP(3330) NVFY VOLID(444444) DVTOC(0,1,1) -
       MIMIC(MINI(10))


3   File: DSF EXEC

/* EXEC to Invoke Device Support Facility */
arg cntrl .
address command
'CP CLOSE READER'
'CP PURGE READER CLASS I'
'CP SPOOL PUNCH CONT TO * CLASS I'
'PUNCH IPL DSF S ( NOH'
'PUNCH' cntrl 'DSF ( NOH'
'CP SPOOL PUNCH NOCONT CLOSE'
'CP SPOOL READER CLASS I NOHOLD'
'CP IPL 00C CLEAR ATTN'
```

----

1 These commands define temporary 3330 minidisks at virtual addresses 200, 300 and 400.

2 This file contains control statements for the Device Support Facility program, which initializes disks for use by VSAM. These disks are labelled 222222, 333333 and 444444.

3 This file contains the commands necessary to use the Device Support Facility program in a virtual machine.

```
4   exec dsf punch

    NO FILES PURGED
    PUN FILE nnnn  TO  CAMPBEL COPY 001   NOHOLD
5   ICK005E DEFINE INPUT DEVICE, REPLY 'DDDD,CUU' or 'CONSOLE'
    ENTER INPUT/COMMAND:

6   2540,00c
    2540,00C
    ICK006E DEFINE OUTPUT DEVICE, REPLY 'DDDD,CUU' or 'CONSOLE'
    ENTER INPUT/COMMAND:

7   console
    CONSOLE
    ICKDSF -  SA DEVICE SUPPORT FACILITIES 5.0  TIME20:26:00  03/09/82   PAGE  1
     INIT UNIT(200) DEVTYP(3330) NVFY VOLID(222222) DVTOC(0,1,1) -
       MIMIC(MINI(10))
    ICK00700I 200 BEING PROCESSED AS LOGICAL DEVICE = 3330
              PHYSICAL DEVICE = 3330-11
    ICK003D REPLY U TO ALTER VOLUME 200 CONTENTS, ELSE T
    ENTER INPUT/COMMAND:

8   u
    U
    ICK01314I VTOC IS LOCATED AT CCHH=X'0000 0001' AND IS     1 TRACKS.
    ICK00001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0

     INIT UNIT(300) DEVTYP(3330) NVFY VOLID(333333) DVTOC(0,1,1) -
       MIMIC(MINI(10))
    ICK00700I 300 BEING PROCESSED AS LOGICAL DEVICE = 3330
              PHYSICAL DEVICE = 3330-11
    ICK003D REPLY U TO ALTER VOLUME 300 CONTENTS, ELSE T
    ENTER INPUT/COMMAND:
    u
    U
    ICK01314I VTOC IS LOCATED AT CCHH=X'0000 0001' AND IS     1 TRACKS.
    ICK00001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0

     INIT UNIT(400) DEVTYP(3330) NVFY VOLID(444444) DVTOC(0,1,1) -
       MIMIC(MINI(10))
    ICK00700I 400 BEING PROCESSED AS LOGICAL DEVICE = 3330
              PHYSICAL DEVICE = 3330-11
```

----

4 Execute the DSF EXEC, specifying that the Device Support Facility control statements contained in the file 'PUNCH DSF' should be appended to the standalone Device Support Facility program.

5 These messages are issued by the Device Support Facility standalone program.

6 Since the Device Support Facility control statements reside in the virtual card reader, you must indicate to Device Support Facility the device type and the address of your virtual reader.

7 This response tells Device Support Facility to output all run time information to your virtual machine console.

8 This response gives Device Support Facility permission to proceed with the initialization of the disk.

```
    ICK003D REPLY U TO ALTER VOLUME 400 CONTENTS, ELSE T
    ENTER INPUT/COMMAND:
    u
    U
    ICK01314I VTOC IS LOCATED AT CCHH=X'0000 0001' AND IS     1 TRACKS.
    ICK00001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0

    ICKDSF MAXIMUM STORAGE USED = 278968 BYTES  (FIXED = 258120,
           DYNAMIC = 020848)
    ICK00002I ICKDSF PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 0
9   cp ipl cms parm autocr
    Ready;
    CMS VM/SP5.0 SL 000
    Ready;

10  cp link vseaf 350 350 rr pass=read
    DASD 350 LINKED R/O; R/W BY GANDALF
    access 350 z
    DMSACC723I Z (350) R/O - DOS
    Ready;
    set dos on z ( vsam
    Ready;

11  access 200 b
    DMSACC723I B (200) R/W - DOS
    Ready;
    access 300 c
    DMSACC723I C (300) R/W - DOS
    Ready;
    access 400 d
    DMSACC723I D (400) R/W - DOS
    Ready;
```

----

9 You must re-IPL CMS after all Device Support Facility processing has completed.

10 If you are a CMS/DOS user, you must access the VSE/AF SYSRES disk and issue the 'SET DOS ON fm (VSAM' command. If you have not previously linked to the VSE/AF SYSRES, you must use the CP LINK command before you issue the ACCESS command. Another method is to have the operator ATTACH the SYSRES disk to your virtual machine. Consult with your system programmer for the procedure to use at your installation.

11 ACCESS the three newly formatted disks as your B-, C- and D-disks.

```
12 query search
    PLC191  191  A    R/W
    222222  200  B    R/W - DOS
    333333  300  C    R/W - DOS
    444444  400  D    R/W - DOS
    MNT190  190  S    R/O
    MNT191  190  Y/S  R/O
    VSERES  350  Z    R/O - DOS
    Ready;

13  File: MASTCAT AMSERV
     DEFINE MASTERCATALOG -
       (  NAME  (MASTCAT)  -
          VOLUME (222222) -
          CYL (4) -
          UPDATEPW (GAZELLE) -
          FILE (IJSYSCT) ) DATA (CYL(1))

14  assgn syscat b
    Ready;
   dlbl ijsysct b dsn mastcat  (syscat perm extent
    DMSDLB331R Enter extent specifications:
    19 171

15
    Ready;

16 amserv mastcat
    Ready;
```

----

12 You can issue the QUERY SEARCH command to verify the status of all disks you currently have accessed. The 350 disk will be listed only if DOS is set on.

13 The file MASTCAT AMSERV defines the VSAM master catalog that you are going to use and provides space for suballocated clusters.

14 Identify the master catalog volume, and use the EXTENT option on the DLBL command so that you can enter the extents. For this extent, specify 171 tracks (9 cylinders) for the master catalog. Since 4 cylinders are specified in the AMSERV file, the remaining 5 cylinders will be used for suballocation by VSAM.

15 You must enter a null line to indicate that you have finished entering extent information.

16 Issue the AMSERV command specifying the MASTCAT file. The ready message indicates that the master catalog is created.

```
17  File: CLUSTER AMSERV
     DEFINE CLUSTER ( NAME (BOOK.LIST  ) -
        VOLUMES (222222) -
        TRACKS (20) -
        KEYS (14,0) -
        RECORDSIZE (120,132) ) -
        DATA (NAME (BOOK.LIST.DATA) ) -
        INDEX (NAME (BOOK.LIST.INDEX ) )

18  amserv cluster
    4221A ATTEMPT 1 OF 2. ENTER PASSWORD FOR JOB AMSERV   FILE MASTCAT
    gazelle
    Ready;

19  File: REPRO AMSERV
      REPRO INFILE (BFILE -
            ENV ( RECORDFORMAT(F) -
            BLOCKSIZE(120) -
            PDEV (3330) ) ) -
            OUTFILE (BOOK)

20  assgn sys001 a
    Ready;
    copyfile test data a (recfm f lrecl 120
    Ready;
    sort test data a book file a
    DMSSRT604R Enter sort fields:
    1 14
    Ready;
    dlbl bfile a cms book file (sys001
    Ready;
21  assgn sys002 b
    Ready;
    dlbl book b dsn book.list (vsam sys002
    Ready;
    amserv repro
    Ready;
```

----

17 Define a suballocated cluster. This cluster is for a key-sequenced data set named BOOK.LIST.

18 No DLBL command is necessary when you define a suballocated cluster. Not that since the password was not provided in the AMSERV file, access method services prompts you to enter the password of the catalog, which is defined as GAZELLE.

19 Use the access method services REPRO command to copy a CMS data file into the cluster that you just defined.

20 You must identify the ddnames for the input and output files for the REPRO function. BFILE is a CMS file, which must be a fixed-length, 120-character file, and it must be sorted alphamerically in columns 1 through 14. The COPYFILE command can copy any existing file that you have to the proper record format; the SORT command sorts the records on the proper fields.

21 The output file is the VSAM cluster, so you must use the VSAM option on this DLBL command.

```
22  File: SPACE AMSERV
      DEFINE SPACE -
           ( FILE (SPACE) -
             TRACKS (57) -
             VOLUME (333333) )


    assgn sys003 c
    Ready;
23  dlbl space c (extent sys003
    DMSDLB331R Enter extent specifications:
    19 57

    Ready;
24  amserv space
    4221A ATTEMPT 1 OF 2. ENTER PASSWORD FOR JOB AMSERV   FILE MASTCAT
    gazelle
    Ready;
25  File: UNIQUE AMSERV
     DEFINE  CLUSTER -
       ( NAME (UNIQUE.FILE) -
         UNIQUE ) -
       DATA  -
       ( CYL (3) -
         FILE (KDATA) -
         RECORDSIZE (100 132) -
         KEYS(12,0) -
         VOLUMES (333333 ) ) -
       INDEX -
       ( CYL  (1) -
         FILE (KINDEX) -
         VOLUMES (333333) )
```

----

22 Create an AMSERV file to define additional space for the master catalog on the volume labelled 333333.

23 Again, use the EXTENT option on the DLBL command so that you can enter extent information and a null line to indicate that you have finished entering extents.

24 Issue the AMSERV command. Again, you are prompted to enter the password of the master catalog.

25 This AMSERV file defines a unique cluster, with data and index components.

```
26  dlbl kdata c (extent  sys003
     DMSDLB331R Enter extent specifications:
     76 57

     Ready;
    dlbl kindex c (extent  sys003
     DMSDLB331R Enter extent specifications:
     76 76

     Ready;
    amserv unique
     4221A ATTEMPT 1 OF 2. ENTER PASSWORD FOR JOB AMSERV   FILE MASTCAT
    gazelle
     Ready;
27.  File: USERCAT AMSERV
      DEFINE USERCATALOG -
           ( CYL (8) -
             FILE (IJSYSUC) -
             NAME (PRIVATE.CATALOG) -
             VOLUME (444444) -
             UPDATEPW (UNICORN) -
             ATTEMPTS (2) ) -
            DATA  (CYL (3)  ) -
            INDEX ( CYL (1) ) -
            CATALOG (MASTCAT/GAZELLE )
28   assgn sys006 d
     Ready;
    dlbl ijsysuc d dsn private.catalog (extent  sys006 perm
     DMSDLB331R Enter extent specifications:
    19 152

     Ready;
    amserv usercat
     *
     Ready;
```

----

26 You must enter DLBL command and extent information for both the data and index components of the unique cluster.

27 Next, define a private (user) catalog for the volume 444444. This catalog is named PRIVATE.CATALOG and has a password of UNICORN. Again, as in step 13, space is made available for suballocation.

28 When you define a user catalog that you are going to use as the job catalog for a terminal session, you should use the ddname IJSYSUC.

```
29  Tape 181 attached

30  File: EXPORT AMSERV
      EXPORT BOOK.LIST  -
         INFILE (BOOK) -
         OUTFILE (TEMP  ENV (PDEV (2400)  REWIND NOLABEL ) )
31   dlbl book b dsn book list (cat ijsysct sys002
    Ready;
32   amserv export (tapout 181
    DMSAMS367R Enter tape output DDNAMEs:
     temp
    Ready;
33  File: IMPORT AMSERV

     IMPORT   -
             CATALOG (PRIVATE.CATALOG/UNICORN) -
             INFILE (TEMP ENV (PDEV (2400) REWIND NOLABEL)) -
             OBJECTS (BOOK.LIST VOL (444444))
34   amserv import (tapin 181
    DMSAMS367R Enter tape input DDNAMEs:
     temp
    Ready;
```

----

29 You may want to try an EXPORT/IMPORT function, if you can obtain a scratch tape from the operator. When the tape is attached to your virtual machine, you receive this message.

30 The file that is being exported is the cluster BOOK.LIST created above. If you do not have access to a tape, you can export the file to you CMS A-disk. Remember to change the PDEV parameter to reflect the appropriate device type.

31 You must reissue the DLBL for BOOK.LIST because there is a job catalog in effect, and the file is cataloged in the master catalog. Use the CAT option to override the job catalog.

32 There is no default tape value when you are using tapes with the AMSERV command. You must specify the TAPIN or TAPOUT option and indicate the virtual address of the tape. You are prompted to enter the ddname, which for this file is TEMP.

33 The last AMSERV file imports the cluster BOOK.LIST to the user catalog PRIVATE.CATALOG.

34 Read the tape in as input.

Structural Changes

This book contains material formerly found in the *VM/SP System Programmer's Guide*, SC19-6203. Figure 40 on page 380 shows the Release 4 books now obsolete by this reorganization, the new system programming books, and the topics these new books contain.

To obtain editions of the *VM/SP System Programmer's Guide*, you must order using the pseudo-number assigned to the respective edition. For:

VM/SP Release 4, order ST00-1578

VM/SP Release 3, order ST00-1352

VM/SP Release 2, order SQ19-6203

VM/SP Release 1, order ST19-6203.

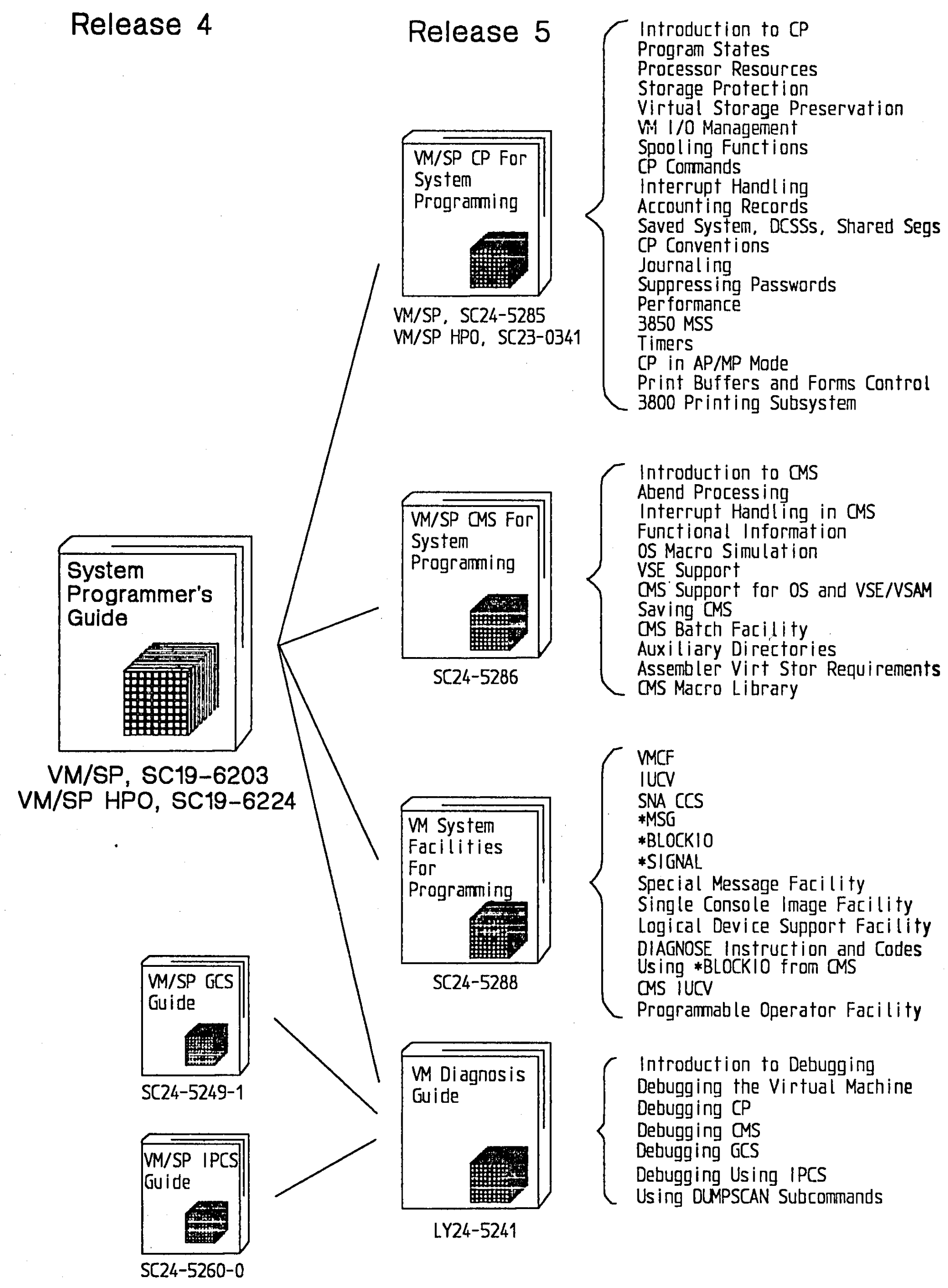


Figure 40. New VM/SP System Programming Manuals for VM/SP Release 5

Technical Changes for VM System Facilities for Programming

Summary of Changes
for SC24-5286-0
for VM/SP Release 5

VM/SP Enhanced Usability

VM/SP now has the following usability enhancements:

- window functions
- full-screen environment for CMS
- a CONSOLE macro.

When CMS is in full-screen 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.

New CMS Commands

SET TRANSLATE Command
The SET TRANSLATE command specifies whether to use the User National Language Translation Tables and/or the System National Language Translation Tables. This command also suppresses translations and translation synonyms of command names for a language.

SET LANGUAGE Command
The SET LANGUAGE command changes the current language of your CMS session and any application running on CMS that uses national language support.

PARSECMD Command
The PARSECMD command invokes the parsing facility from an EXEC.

New CMS Macros

All new CMS macros are listed in Appendix A, "CMS Macro Library" on page 351.

Modified CMS Commands

GLOBAL Command
The enhanced GLOBAL command 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.

TXTLIB Command
The FILENAME option of the TXTLIB command creates a directory entry in the TXTLIB for each filename of the TEXT files specified.

LOAD Command
The HIST option of the LOAD command lets you add comments from TEXT files into MODULE files.

INCLUDE Command
The HIST option of the LOAD 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 Segement (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.

Creating Message Files

VM/SP now lets you store any message text in one central file. This file is called a respository file. Some advantages of having a repository file are:

- message text will not clutter your program

- message text can be translated to another language easily

Enhanced Connectivity Facilities on VM/SP

This part of CMS supports IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities on a VM/SP system. For detailed information about Enhanced Connectivity Facilities on VM/SP, refer to the *IBM VM/SP Programmer's Guide to the Server-Requester Programming Interface for VM/SP*, SC24-5291.

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 the VM/SP System Programmer's Guide

The following "Summary of Changes" reflect the changes made to the *VM/SP System Programmer's Guide*, Release 4 and Release 3.

Summary of Changes
for SC19-6203-3
for VM/SP Release 4

Group Control System (VM/SP GCS)

This new component of VM/SP is a virtual machine supervisor that provides simulated MVS services and supports a multitasking environment. For more information on the Group Control System (GCS), refer to the *VM/SP Group Control System Guide - (Discontinued)*, SC24-5249.

Signal System Service

This new CP system service allows virtual machines in a Virtual Machine Group to signal each other. The Signal System Service can only be used by virtual machines in a Virtual Machine Group.

Saved System 8M Byte Limit Removal

With the addition of this support, the SAVESYS, VMSAVE, and IPL functions have been enhanced to allow a page image copy of up to a 16M byte virtual machine to be saved and restored.

CP FRET Trap

The CP FRET Trap can be used as an aid in solving problems caused by improper use of CP storage and to solve many storage overlay problems.

VMDUMP Enhancements

DIAGNOSE code X'94' is available to allow a virtual machine to request dumping of its virtual storage. Also, the three address range restriction has been removed from the VMDUMP command.

DIAGNOSE code X'98'

Using DIAGNOSE code X'98' a virtual machine can lock and unlock virtual pages, and execute its own real channel programs.

The Programmable Operator Facility

The Programmable Operator Facility has been enhanced to support distributed operations in an SNA network through an interface, the Programmable Operator/NCCF Message Exchange (PMX), with the Network Communications Control Facility (NCCF). The VM/SP Release 4 programmable operator:

- Allows an NCCF operator to be identified to the programmable operator so that any messages intended for the logical operator may be routed to that NCCF operator.
- Allows an NCCF operator to issue programmable operator commands and receive responses.
- Provides the LGLOPR command for assigning, releasing and replacing the logical operator during operation.

CPTRAP Enhancements

CPTRAP is a major service aid used in problem determination. Enhancements to the CPTRAP command provide two additional functions, GROUPID and WRAP, and one additional entry type, X'3D'.

Enhancements to TRAPRED makes reviewing the trap data easier by providing more selectivity for X'3D', X'3E', and X'3F' entries and by providing a way to display formatted output of the trapped data.

Information on CPTRAP has been rewritten and reorganized for ease-of-use. It has also been moved to the Part 3, the debugging section, since it is a debugging tool.

Interactive Problem Control System (VM/SP IPCS)

VM/SP Release 4 has been enhanced to include IPCS as a component of VM/SP. VM/SP IPCS is equivalent to the VM/Interactive Problem Control System Extension (VM/IPCS/E) Program Product (5748-SA1).

Inter-User Communications Vehicle (IUCV) Enhancements

IUCV now supports the movement of data on the SEND, RECEIVE, and REPLY functions from discontiguous buffers. The modified IUCV macro handles the new BUFLIST= parameter on SEND and RECEIVE functions and the new ANSLIST= parameter on the SEND and REPLY functions.

Expansion of User Classes

The DIRECT command has been enhanced and the OVERRIDE command has been added to provide the user with more than the seven IBM defined user classes. You can now choose from 32 user classes, A - Z, and 1 - 6.

Remote Spooling Communications Subsystem Networking Version 2

With the release of the Remote Spooling Communications Subsystem Networking Version 2 Program Product (5664-188), any reference to RSCS in this manual applies to RSCS Version 2. Information pertaining to RSCS can be found in the *VM/SP Remote Spooling Communications Subsystem Version 2 General Information*, GH24-5055.

Miscellaneous

IOCP Support Enhancements

This support adds new MSSF command words to DIAGNOSE code X'80'.

Integration of Functional Enhancements to VM/SP Release 3

Information has been added to support:

- The 3290 Information Panel
- The 3370 Direct Access Storage Model
- The 4248 Printer
- The 4361 Model Groups 3, 4, and 5 Processor
- The 4381 Model Groups 1 and 2 Processor
- VM/SP 3800 Model 3 Compatibility Support

  Compatibility support allows VM/SP users to access the 3800 Model 3 Printing Subsystem. Existing programs designed to produce 3800 Model 1 printer output may produce output for the 3800 Model 3 printer with little or no program change. Use of this support provides improved print quality (240 x 240 pel resolution) and the addition of a 10 lines-per-inch (LPI) vertical space option.

DIAGNOSE code X'8C'

DIAGNOSE code X'8C' has been enhanced to allow a user to access all of the data returned by CP's WRITE STRUCTURED FIELD QUERY.

DMKFRE/DMKFRT Split

The module DMKFRE has been split into two modules, DMKFRE and DMKFRT. DMKFRE handles all requests for free storage as well as calls to DMKFRET to release free storage. DMKFRT handles all requests to return free storage that cannot be handled by the microcoded CP assist FRET function.

Minor technical and editorial changes have been made throughout this publication.

Summary of Changes
for SC19-6203-2
for VM/SP Release 3

Programmable Operator Facility

Several enhancements to the programmable operator facility added are:

- Message routing with nicknames
- Remote node availability
- Enhanced text comparison
- EXEC action routines
- LOG recording and error handling

PER

Problem determination capability is greatly extended and enhanced by the new CP command, PER.

DASD Block I/O System Service

The DASD Block I/O System Service allows a virtual machine fast, device-independent asynchronous access to fixed size blocks on CMS formatted virtual DASD I/O devices.

IUCV

Inter-User Communication Vehicle (IUCV) extensions provide:

- SEND and REPLY extensions
- An extended mask capability for control interrupts
- An expanded trace capability to record all IUCV operations
- A macro option to initialize the parameter list
- Support for the DASD block I/O system service.

The IBM 3088 Multisystem Communications Unit

The IBM 3088 Multisystem Communications Unit interconnects multiple systems using block multiplexer channels. The 3088 uses an unshared subchannel for each unique address and is fully compatible with existing channel-to-channel adapter protocol.

CMS IUCV Support

Support for IUCV communication has been introduced into CMS. This support allows multiple programs within a virtual machine to use IUCV functions. Included is the ability to initialize a CMS machine for IUCV communication and to invoke IUCV functions via new CMS macros. These macros also allow the user to specify path-specific exits for IUCV external interrupts.

CMS Abend Exits

A general CMS abnormal exit capability is provided so that user programs may specify the address of a routine to get control before CMS abend recovery begins. An exit is established and cleared through a new CMS macro.

Enhanced Immediate Command Support

The immediate command capability of CMS is extended by allowing users to define their own immediate commands.

Enhanced VSAM Support

CMS supports VSE/VSAM Release 3 which includes significant enhancements designed to improve catalog reliability and integrity while providing additional serviceability and usability. VSE/VSAM Release 2 is not supported.

Miscellaneous

Changes to the DIAGNOSE code X'00' interface provide the time zone differential from Greenwich Mean Time.

DIAGNOSE code X'8C' allows a virtual machine to access device dependent information without having to issue a WRITE STRUCTURE FIELD QUERY REPLY.

CMSSEG has been eliminated and the code was merged into the CMS Nucleus.

The Remote Spooling Communications Subsystem (RSCS) section of this manual has been removed as it pertained to RSCS as a component of VM/370. Now, any reference to RSCS in this manual applies to the RSCS Networking Programming Product, and information can be found in the *VM/SP Remote Spooling Communications Subsystem Networking Program Reference and Operations Manual*, SH24-5005.

A newly added appendix lists and describes the CMS macros applicable to VM/SP.

Minor technical and editorial changes have been made throughout this publication.

Abbreviations

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

Unless otherwise noted,

VM/SP	refers to the VM/SP program package when you use it in conjunction with VM/370 Release 6.
CP	refers to the VM/370 Control Program component enhanced by the functions included in the VM/SP package.
CMS	refers to the VM/370 Conversational Monitor System component enhanced by the functions included in the VM/SP package.
GCS	refers to the Group Control System component of VM/SP. See the *Group Control System Command and Macro Reference*, SC24-5250, for details of GCS.
IPCS	refers to the VM/370 Interactive Problem Control System component enhanced by the functions included in the VM/SP package. The IPCS component of VM/SP replaces the unmodified VM/370 interactive problem control system. Details describing this component are found in the *VM Diagnosis Guide*, LY24-5241.
RSCS	unless otherwise noted, refers to the RSCS Networking Version 2 Program Product (5664-188). When you install and use VM/SP in conjunction with the VM/370 Release 6 System Control Program (SCP), it becomes a functional operating system that provides extended features to the Control Program (CP) and Conversational Monitor System (CMS) components of VM/370 Release 6. VM/SP adds *no* additional functions to the Remote Spooling Communications Subsystem (RSCS) component of VM/370. However, you can appreciably expand the capabilities of this component in a VM/SP system by installing RSCS Networking Version 2 (5664-188).
VSE	refers to the combination of the DOS/VSE system control program and the VSE/Advanced Functions Program Product. "DOS", in certain cases, is still used as a generic term. For example, disk packs initialized for use with VSE or any predecessor DOS or DOS/VSE system may be referred to as DOS disks.
CMS/DOS	refers to the DOS-like simulation environment provided under the CMS component of the VM/SP.
EXEC	refers to EXECs using the System Product Interpreter (REXX), EXEC 2, or CMS EXEC languages.
System/370	applies to the 4300 and 303X series of processors.

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

3066	refers to the IBM 3066 System Console.
3088	refers to the IBM 3088 Multisystem Communications Unit (MCU) Models 1 and 2.
3262	refers to the IBM 3262 Printer, Models 1, 5, and 11. 3262 Models 3 and 13 are supported remotely as 3287 printers.
3270	refers to a series of display devices, namely, the IBM 3275, 3276 (referred to as a Controller Display Station), 3277, 3278, and 3279 Display Stations, and the 3290 Information Panel. A specific device type is used only when a distinction is required between device types. Information about display terminal use also applies to the IBM 3138, 3148, and 3158 Display Consoles when used in display mode, unless otherwise noted.

3285	or 3286 printer references also pertain to the IBM 3287, 3288, and 3289 printers, unless otherwise noted.
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 compatibility mode.
3340	refers to the IBM 3340 Direct Access Storage Facility and the 3344 Direct Access Storage.
3350	refers to the IBM 3350 Direct Access Storage Device when used in native mode.
3370	refers to the IBM 3370 Direct Access Storage Model.
3375	refers to the IBM 3375 Direct Access Device.
3380	refers to the IBM 3380 Direct Access Storage. The Speed Matching Buffer Feature (No. 6550) for the 3380 supports the use of extended count-key-data channel programs.
3422	refers to IBM 3422 Magnetic Tape Subsystem.
3480	refers to the IBM 3480 Magnetic Tape Subsystem.
3430	refers to the IBM 3430 Magnetic Tape Subsystem.
3800	refers to the IBM 3800 Printing Subsystems, Models 1, 3, and 8. A specific device type is used only when a distinction is required between device types. References to the 3800 Model 3 apply to both Models 3 and 8 unless otherwise explicitly stated. The IBM 3800 Model 8 is available only in selected world trade countries.
4245	refers to the IBM 4245 Line Printer.
4248	refers to the IBM 4248 Printer.
4250	refers to the IBM 4250 Printer.
4361	refers to the IBM 4361 Model Groups 3, 4, and 5 Processor.
4381	refers to the IBM 4381 Model Groups 1 and 2 Processor.

Glossary

This glossary defines new terms and all-capital abbreviations related to the VM/SP. This glossary is especially oriented for readers of the *VM/SP CMS for System Programming*. Therefore, some terms already defined in the *VM/SP Library Guide, Glossary, and Master Index*, SC19-6207, do not appear here or may be defined slightly differently. | Another glossary you may refer to is the *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems.*

A

auxiliary storage. Data storage other than main storage; in VM/SP, auxiliary storage is usually a direct access device.

C

CAW. channel address word

CCW. channel command word

channel address word (CAW). An area in storage that specifies the location in main storage at which a channel program begins.

channel command word (CCW). A doubleword at the location in main storage specified by the channel address word. One or more CCWs make up the channel program that directs data channel operations.

channel status word (CSW). An area in storage that provides information about the termination of input/output operations.

Channel-to-Channel Adapter. A hardware device that can be used to connect two channels on the same computing system or on different systems.

CKD. Count-Key-Data

concurrently. Concerning a mode of operation that includes the performance of two or more operations within a given interval of time.

| **CMS system disk.** The virtual disk (S-disk) that contains the CMS nucleus and the disk-resident CMS commands. The CMS system disk can have extensions, usually the Y-disk.

Count-Key-Data. Those DASD devices whose architecture defines variable size records consisting of count, key, and data fields.

CSW. channel status word

D

DAT. dynamic address translation.

DCSS. discontiguous shared segments.

deadline priority. An algorithm for determining when a virtual machine receives the next time slice.

directory. For VM/SP, a CP disk file that defines each virtual machine's normal configuration: the userid, password, normal and maximum allowable virtual storage, CP command privilege class or classes allowed, dispatching priority, logical editing symbols to be used, account number, and CP options desired.

discontiguous shared segments (DCSS). Synonymous with *discontiguous segment.*

discontiguous segment. A 64K segment of storage that was previously loaded and saved and assigned a unique name. The segment(s) can be shared among virtual machines if the segment(s) contain reentrant code.

DPA. dynamic paging area

dynamic address translation. In System/370 virtual storage systems, the change of a virtual address to a real storage address during execution of an instruction.

dynamic paging area (DPA). An area of real storage that CP uses for virtual machine pages and pageable CP modules.

F

FBA. Fixed-block architecture.

| **file status table (FST).** A table that describes the attributes of a file on a CMS disk, including filename, filetype, filemode, date last written, and other status information.

Fixed-Block Architecture (FBA). Those DASD devices whose architecture uses fixed blocks or records of 512 bytes.

| **FST.** file status table

G

| **GCS.** Group Control System facility

| **Group Control System.** An operating environment that provides a problem state OS subtasking environment with common storage access for members of a virtual machine group.

guest virtual machine. A virtual machine in which an operating system is running.

I

in-queue virtual machines. A virtual machine on the run list waiting to be dispatched. A virtual machine is added to the run list if its projected working set size is less than or equal to the number of real page frames available for allocation in the dynamic paging area. An in-queue virtual machine may be, but is not necessarily, runnable.

interactive. (1) An application in which each user entry calls forth a response from a system or program. (2) The classification given to a virtual machine depending on this virtual machine's processing characteristics. When a virtual machine uses less than its allocated time slice because of terminal I/O, the virtual machine is classified as being interactive. See also non-interactive.

Interactive Problem Control System (IPCS or VM/SP IPCS). A component of VM/SP that permits on-line problem management, interactive problem diagnosis, on-line debugging for disk-related CP or virtual machine abend dumps, problem tracking, and problem reporting.

L

logon. The procedure by which a user begins a terminal session.

logoff. The procedure by which a user ends a terminal session.

M

minidisk. Synonym for virtual disk.

missing interrupt handler (MIH). A facility of VM/SP that detects incomplete I/O conditions by monitoring I/O activity. It also tries to correct incomplete I/O conditions without operator intervention.

N

named system. A collection of saved pages a user can IPL or load by name.

native mode. A mode in which an operating system is run stand-alone on the real machine instead of under VM/SP.

noninteractive. The classification given to a virtual machine depending on this virtual machine's processing characteristics. When a virtual machine usually uses all its allocated time slice, it is classified as being noninteractive or compute bound. See also interactive.

non-resident pages. Pages whose contents are on DASD but not in real storage. A page is considered nonresident when an attempt to load its real address returns a nonzero condition code.

P

page frame. A block of 4096 bytes of real storage.

page table. A table in CP that indicates whether a page is in real storage and matches virtual addresses with real storage addresses.

preferred paging area. A special area of auxiliary storage where frequently used pages are paged out. It provides high speed paging.

prefix storage area (PSA). a page zero of real storage that contains machine-used data areas and CP global data.

program status word (PSW). An area in storage used to indicate the order in which instructions are executed, and to hold and indicate the status of the computer system. Synonymous with processor status word.

program temporary fix (PTF). A temporary solution or by-pass of a problem diagnosed by IBM field engineering as the result of a defect in a current unaltered release of the program.

PSA. Prefix storage area.

PSW. Program status word, or processor status word.

PTF. Program temporary fix.

Q

queue-add. The action by the system scheduler, DMKSCH, of placing a runnable virtual machine on the list of virtual machines that can be given control of a processor.

queue-drop. The action by the system scheduler, DMKSCH, of removing a virtual machine from the list of virtual machines that can be given control of a processor.

R

real machine. The actual processor, channels, storage, and I/O devices required for operation of VM/SP.

S

| **S-disk.** See CMS system disk.

| **S-STAT.** A block of storage that contains the file status tables (FSTs) associated with the S-disk. The FSTs are sorted so that a binary search can be used to search for files. The S-STAT usually resides in the CMS nucleus so it can be shared. Only files with filemode of 2 will have their associated FSTs in the S-STAT.

segment. A contiguous 64K area of virtual storage (not necessarily contiguous in real storage) that is allocated to virtual machine or CP.

segment table. A table used in dynamic address translation to control user access to virtual storage segments. Each entry indicates the length, location, and availability of a corresponding page table.

shadow page table. A table that maps real storage allocations (first level storage) to a virtual machine's virtual storage (third level storage) for use by the real machine in its paging operations.

spool, spooled, spooling. Relates to the reading of input data streams and the writing of output data streams on auxiliary storage devices.

standalone dump. A program used to print the contents of storage that runs in a virtual machine not under control of an operating system such as CMS.

| **system profile.** The system profile is an EXEC, SYSPROF EXEC, that resides in a DCSS (discontiguous saved segment) or on a system disk and is called by CMS initialization. It contains some initialization function and provides a means for installations to override the default CMS environment by tailoring the EXEC to suit the installation.

T

time sharing. Sharing of computer time and resources.

V

virtual address. An address that refers to virtual storage or a virtual I/O device address. It must, therefore, be translated into a real storage or I/O device address when it is used.

virtual disk. A logical subdivision (or all) of a physical disk storage device that has its own address, consecutive storage space for data, and an index or description of the stored data so that the data can be accessed. A virtual disk is also called a minidisk.

virtual machine. A functional simulation of a computer and its associated devices.

Virtual Machine Communication Facility (VMCF). A CP function that provides a method of communication and data transfer between virtual machines operating under the same VM/SP systems.

virtual storage. Storage space that can be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computing system and by the amount of auxiliary storage available, and not by the actual number of main storage locations.

Y

Y-disk. An extension of the CMS system disk.

Y-STAT. A block of storage that contains the file status tables (FSTs) associated with the Y-disk. The FSTs are sorted so that a binary search can be used to search for files. The Y-STAT usually resides in the CMS nucleus so it can be shared. Only files with filemode of 2 will have their associated FSTs in the Y-STAT.

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*, GC20-0001.

- **Prerequisite Publications**

  IBM System/360 Principles of Operation, GA22-6821

  IBM System/370 Principles of Operation, GA22-7000.

- **Books About VM/SP**

  Virtual Machine/System Product:

  CP for System Programming, SC24-5285

  Transparent Services Access Facility Reference, SC24-5287

  General Information, GC20-1838

  Introduction, GC19-6200

  CMS Command Reference, SC19-6209

  CMS User's Guide, SC19-6210

  Installation Guide, SC24-5237

  System Messages and Codes, SC19-6204

  OLTSEP and Error Recording Guide, SC19-6205

  Terminal Reference, SC19-6206

  Library Guide, Glossary, and Master Index, SC19-6207

  Operator's Guide, SC19-6202

  EXEC 2 Reference, SC24-5219

  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

  Virtual Machine

  System Facilities for Programming, SC24-5288

Diagnosis Guide, LY24-5241

Running Guest Operating Systems, SC19-6212

Note: The *VM/SP Library Guide, Glossary, and Master Index,* GC19-6207 describes all the VM/SP books and contains an expanded glossary and master index to all the books in the VM/SP library.

- **Other Publications**

IBM VM/SP Programmer's Guide to the Server-Requester Programming Interface for VM/SP, SC24-5291

IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide, GA24-3543

IBM 3262 Printers 1 and 11 Component Description, GA24-3733

IBM 3270 Information Display System Library User's Guide, GA23-0058

IBM Virtual Machine Facility/370: Performance/Monitor Analysis Program, SB21-2101

ACF/VTAM

VTAM General Information (for VM), GC30-3246

Network Program Products Planning, SC23-0110

VTAM Installation and Resource Definition, SC23-0111

VTAM Customization, SC23-0112

VTAM Operation, SC23-0113.

VM/SP Remote Spooling Communications Subsystem Networking (RSCS Networking) Version 2

Planning and Installation, SH24-5057

Operation and Use, SH24-5058

Diagnosis Reference, LY24-5228

VM/SP Data Areas and Control Block Logic,

Volume 2 Conversational Monitor System (CMS), LY24-5221

VM/SP System Logic and Problem Determination,

Volume 2 Conversational Monitor System (CMS), LY20-0893

OS/VS Data Management Macro Instructions, GC26-3793

OS/VS Supervisor Service and Macro Instructions, GC27-6979

If you use the IBM 3767 Communication Terminal as a virtual machine console, the *IBM 3767 Operator's Guide,* GA18-2000 may also be helpful.

Note: References in text to titles of corequisite VM/SP Entry and VM/SP publications are given in abbreviated form.

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

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
SFPROG—VM System Facilities for Programming
DIAG—VM Diagnosis Guide

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
DIAG—VM Diagnosis Guide
SFPROG—VM System Facilities for Programming

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
SFPROG—VM System Facilities for Programming
DIAG—VM Diagnosis Guide

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
SFPROG—VM System Facilities for Programming
DIAG—VM Diagnosis Guide

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
SFPROG—VM System Facilities for Programming
DIAG—VM Diagnosis Guide

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
DIAG—VM Diagnosis Guide
SFPROG—VM System Facilities for Programming

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
DIAG—VM Diagnosis Guide
SFPROG—VM System Facilities for Programming

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
SFPROG—VM System Facilities for Programming
DIAG—VM Diagnosis Guide

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG–VM/SP CP for System Programming
SFPROG–VM System Facilities for Programming
DIAG–VM Diagnosis Guide

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
SFPROG—VM System Facilities for Programming
DIAG—VM Diagnosis Guide

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
SFPROG—VM System Facilities for Programming
DIAG—VM Diagnosis Guide

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
SFPROG—VM System Facilities for Programming
DIAG—VM Diagnosis Guide

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
SFPROG—VM System Facilities for Programming
DIAG—VM Diagnosis Guide

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming SFPROG—VM System Facilities for Programming
DIAG—VM Diagnosis Guide

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
DIAG—VM Diagnosis Guide
SFPROG—VM System Facilities for Programming

These symbols are used in the index to refer to other VM and VM/SP books:
CPPROG—VM/SP CP for System Programming
SFPROG—VM System Facilities for Programming
DIAG—VM Diagnosis Guide

International Business
Machines Corporation
P.O. Box 6
Endicott, New York 13760

File No. S370/4300-36
Printed in U.S.A.

SC24-5286-0

VM/SP
CMS for System Programming
Order No. SC24-5286-0

READER'S COMMENT FORM

Is there anything you especially like or dislike about this book? Feel free to comment on specific errors or omissions, accuracy, organization, or completeness of this book.

If you use this form to comment on the online HELP facility, please copy the top line of the HELP screen.

____________ ____________ _______ Help Information line ___ of ___

Note: Do not use this form to report system problems or to request copies of publications. Instead, contact your IBM representative or the IBM branch office serving you.

Would you like a reply? ___YES ___NO

Please print your name, company name, and address:

____

____

____

____

IBM Branch Office serving you: ____________________________________

Thank you for your cooperation. You can either mail this form directly to us or give this form to an IBM representative who will forward it to us.

SC24-5286-0

Reader's Comment Form

CUT
OR
FOLD
ALONG
LINE

Fold and tape   Please Do Not Staple   Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL   PERMIT NO. 40   ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE:

INTERNATIONAL BUSINESS MACHINES CORPORATION
DEPARTMENT G60
PO BOX 6
ENDICOTT NY 13760-9987

Fold and tape   Please Do Not Staple   Fold and tape

®

VM/SP
CMS for System Programming
Order No. SC24-5286-0

READER'S COMMENT FORM

Is there anything you especially like or dislike about this book? Feel free to comment on specific errors or omissions, accuracy, organization, or completeness of this book.

If you use this form to comment on the online HELP facility, please copy the top line of the HELP screen.

______________ ______________ ________ **Help Information line ____ of ____**

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you, and all such information will be considered nonconfidential.

Note: Do not use this form to report system problems or to request copies of publications. Instead, contact your IBM representative or the IBM branch office serving you.

Would you like a reply? ___YES ___NO

Please print your name, company name, and address:

____

____

____

____

IBM Branch Office serving you: ______________________________

Thank you for your cooperation. You can either mail this form directly to us or give this form to an IBM representative who will forward it to us.

SC24-5286-0

Reader's Comment Form

CUT
OR
FOLD
ALONG
LINE

Fold and tape   Please Do Not Staple   Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL   PERMIT NO. 40   ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE:

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. S370/4300-36
Printed in U.S.A.

SC24-5286-0

SC24-5286-00