Virtual Memory Operating System (VMOS)

System Management Reference Manual

May 1972
DP-005-2-00

Series 70 Publications
Building 214-1
Cinnaminson, N.J. 08077

SPERRY RAND
UNIVAC
SERIES 70

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

First Printing: May 1972 (DP-005-2-00)

GENERAL

The VMOS System Management Reference Manual provides systems personnel with the information necessary to manage the throughput of work run under the Virtual Memory Operating System (VMOS). This publication contains the information necessary to use the control system command and macro language (the Command Language) for task, file, and device management.

No attempt is made to prescribe the duties of those systems personnel who will need to use the information presented herein, for it is assumed that the material in this book will frequently be necessary for programming functions as well as system analysis. Also, the areas of source program input, object code generation, the use of diagnostic routines, etc., which relate to program preparation, have been omitted. These subjects are covered in detail in the VMOS Programmer's Reference Manual.

The VMOS System Management Reference Manual has been organized as follows:

Part 1. VMOS System Management. A two-section discussion of the design concepts and the operational characteristics of VMOS as they affect system management comprises this part. Such details are provided by an overview of VMOS, and of the three major VMOS Control Program components which most affect system management.

Part 2. Control Program Usage Concepts. The five sections of this part provide details pertaining to VMOS control program usage in the area of system management. The Introduction section describes certain VMOS concepts relating to program classes, tasks, files, devices, and the use of volumes, that are basic to the effective use of this system. This is followed by a description of the facilities made available for task and program initiation, regulation, and termination; and a discussion of process interruption, program to program communication, remote batch processing, procedure file construction, and checkpoint-restart.

In addition, the concept of VMOS files and access methods; the presentation for device and space allocation, regulation, and deallocation; and information pertaining to communication facilities offered by VMOS are included in this part.

Part 3. Control System Commands and Macros. The four sections that constitute this part contain definitive descriptions of the control system commands and macros relating to system management, including those pertaining to the use of Communications Access Method (CAM).

The VMOS System Management Reference Manual also includes appendices which may be referred to for related material, detailed point description/explanation, extensive test data, tabulations, etc.

REQUISITE READING

The user of this publication is assumed to be a programmer having knowledge of third generation computing equipment and Assembly Language programming. In addition, the reader should familiarize himself with the material in the following publications:

Assembler System Reference Manual,

VMOS Service Routines Manual, and/or

VMOS Programmer's Reference Manual.

CONTENTS

Part 1
VMOS System Management

GENERAL

The management of work performed using the Virtual Memory Operating System (VMOS) is the function of the operating system's Control Program. The various modules which constitute the Control Program have been collected into three functional categories: the Central Control System (Executive), the Data Management System (DMS), and the Language Systems Support (LSS). The processing of work within VMOS is implemented through communication with the Control Program and the medium for this communication is the Command Language.

The terms Executive, DMS, and LSS, while useful as collective terms to describe the functions of modules they comprise, are not easily relatable to the functions a programmer performs in the context of system management. They are used in this manual only as a means of introducing the reader to the Design Concepts of VMOS. In Parts 2 and 3 of this manual the terms task management, file management, and device management are used instead since they more nearly describe the scope of system management. Moreover, these terms are not component oriented but serve to tie together the functions of various components.

The two sections constituting this part provide the user with an overview of the systems, both from a structural point of view and from the standpoint of the operation of the control program. The first section describes the two principle constituents of VMOS: The Control Program and Software-Supplied Programs. Here, the functions of the Executive, DMS, and LSS are described along with the Software-Supplied Service Routines, Language Processors, and Interactive Facilities.

Section 2 describes the functions of the Control Program that most affect system management activities, that is, spoolout, remote terminal input-output, program management, and the operation of the data management system.

GENERAL

Structurally, VMOS can be viewed as consisting of two functionally different facilities: the Control Program and Software-Supplied Processing Routines. The Control Program operates in the privileged modes of the processor and cannot be directly accessed by the user-programmer. VMOS, through the medium of a Command Language, provides a programmer with the means of communicating with the control program.

The Software-Supplied Programs run in the nonprivileged mode of the processor, and are directed by the Control Program to interface with the various users of the system. These routines are called into service by the programmer using the Command Language.

CONTROL PROGRAM

The VMOS Control Program consists of the Central Control System (Executive), Data Management System (DMS), and Language Systems Support (LSS) functional categories. Although these classes are discussed as independent entities, they are interdependent and function as an integrated unit in the processing environment. Descriptions of these functional categories are given below.

The Central Control System (Executive)

The VMOS Executive manages the total operating system environment. It serves as the interface among all users of the system, the computer hardware, and all remaining system software. As far as the user is concerned, the Executive appears to be an integral part of the computer.

The Executive's most frequently used components are permanently resident in physical memory. Other routines are quickly available from the paging drum. The Executive's locations are not addressable by other programs, it executes in the privileged mode and is not generally time-sliced.

The Executive receives interrupts, interprets them, sorts them as to type and function, and initiates the appropriate routines to respond to each. It spools all tasks introduced to the processor to a common task queue.

By means of the time slicing, it provides rapid and complete service for all these tasks, up to the maximum system resource level. The Executive routines control command language processing, system error analysis and recovery functions, job control and system file management, task and memory management, Central Processing Unit (CPU) time, the console and the paging drum, and peripheral device error recovery.

To facilitate scheduling in the system and to ensure optimum processor utilization, a task schedule queue structure has been established consisting of thirteen separate queues. Tasks are placed in appropriate queues according to their needs and conditions and moved among the queues according to the algorithm of the task schedules.

The Executive controls the allocation and utilization of virtual memory, physical memory, and the paging drum backing memory. Although most of this memory management is invisible to the user, it provides the facilities which allow each Class II task in the system to operate as if it had at its disposal up to 256 pages (1,048,576 bytes) of core memory. These pages are contiguously addressed so that the Virtual Memory of each user contains address 0 to 1,048,575.

The upper limit of this virtual memory is dynamically adjusted by the Executive during each task's execution depending on the memory requirements of the task. The information contained in each user's virtual memory is automatically placed into and removed from core storage as needed by the Executive paging algorithm without user knowledge or intervention.

The Executive collects over 20 categories of accounting statistics for each task during its execution. This accounting file is maintained on an on-line disc which may be interrogated by all users of the system.

The Data Management System (DMS)

DMS is the component of the Control Program that performs input/output for the Control Program and user programs. DMS also controls the creation of and access to all data files maintained "on-line" within the system.

The File Management portion of DMS is concerned with cataloging and managing files. File cataloging and management facilities provide the means for identifying files, for storing and retrieving them, for sharing them among users, and for defining their existence and use in the system. File management commands (a subset of the VMOS command language) make the file cataloging and management facilities available to a conversational user or a batch job. These same facilities are available to a user program through the use of file management macros.

The Data Management portion of DMS provides for the actual transfer of data to and from programs that are being processed. Data Management provides several access methods. These access methods provide a simple and efficient means to organize data into a logical structure and then to transfer data between an input/output device and main memory.

Device Management is the I/O dispatcher or device driver for VMOS Device management also provides for the allocation of all I/O devices, with the exception of remote terminals, maintains a list of the devices that are currently available to the system, and provides tasks with these devices as required for I/O processing. Tasks acquire devices for files through the Executive SECURE command the DMS FILE command.

DMS also provides a facility for processing a set of TOS commands and macros. This facility supports file processing and device management for TOS/TDOS (Class I) programs.

Language Systems Support (LSS)

The third major component of the Control Program, Language Systems Support (LSS), contains the systems message processing logic, the system loaders, Desk Calculator Logic, and the Interactive Debuggging Aids (IDA) component.

The VMOS system loaders are the Static Loader and the Dynamic Linking Loader. One or the other is invoked when a programmer calls for a program to be loaded and/or executed. The system chooses the correct loader based on the characteristics of the modules to be processed. These loaders are discussed in detail in the VMOS Service Routines Manual.

The VMOS Message Processing Routine makes it possible for the programmer to create and update a dictionary of encoded messages. These messages can then be called by code number using assembly language programs and sent to the operator or the user. Message processing is discussed in detail in the VMOS Service Routines Manual.

The VMOS Desk Calculator facility enables the programmer to operate his interactive terminal as if it were an electro-mechanical calculator. This facility does not require that the user have programming experience. A full discussion of the Desk Calculator Facility is contained in the VMOS Interactive Language Manual – Desk Calculator.

The VMOS Interactive Debugging Aid (IDA) provides the programmer with the capability to debug object modules either interactively or in the background. IDA permits programmers to check the progress of programs during execution, modify them, and localize trouble areas. A complete discussion of this facility is found in the VMOS Interactive Language Manual – IDA.

SOFTWARE-SUPPLIED PROGRAMS

The second of the two major VMOS facilities, Software-Supplied Programs, comprises the System's Service Routines, the Language Processors, and the Interactive Routines. These are programs which provide a major portion of the system's operational flexibility.

VMOS Service Routines

The VMOS Service Routines comprise a set of Library Utilities, Loading Utilities, Processing Utilities, Display Utilities, and Peripheral to Peripheral Utilities. These utilities are described in detail in the VMOS Service Routines Manual.

Library Utilities

The VMOS Library Utilities provide the user with the capability to create, store, and update source programs, object modules, program load files and macro libraries. They also provide the user with the ability to reformat a complete library or extract a portion thereof in such a manner that the reformated program may be used by TOS or VMOS. The library utilities consist of eight routines as follows:

The Library Maintenance Routine (LMR) creates and maintains object module libraries. LMR operates as a Class II program which may be executed in either the conversational or nonconversational mode.

The Macro Library Update (MLU) routine creates and maintains Assembly language macros. MLU operates as a Class II program.

The TOS Load Module Transcriber routine (TOSLMTP) transcribes load modules of specified programs from the program system load library tape (PLLT/SLLT) of TOS to the program system load library of VMOS. TOSLMTP does this in such a way that the transcribed programs can be used directly by VMOS as Class I programs.

The VMOS Load Module Transcriber routine transcribes files of specified programs residing on disc to the format of the program load library of TOS (on tape) in such a manner that the transcribed programs may be used directly by TOS.

The VMOS COBOL Library to TOS COBOL Library Conversion routine converts a VMOS COBOL library to a TOS COBOL library.

The VMOS OML to CLT Conversion Routine converts a library from VMOS Library Maintenance Routine (LMR) format to TOS Call Library (CLT) format.

The VMOS Assembler Source Language Update facilityis an extension of the Assembler that provides the programmer with the capability for storing and maintaining assembly language source programs on magnetic tape and disc.

The COBOL Library Update Routine consists of 12 commands which enable the user to develop and maintain a COBOL library.

Loading Utilities

The Loading utilities comprise the VMOS Linkage Editor Routine and the VMOS Loaders. These routines prepare and load programs for execution by the user. Note that the loaders operate under direct control of the Control Program. They are summarized in the Control Program portion of this chapter.

The VMOS Linkage Editor routine is a program that forms part of the processing of source programs that follows compilation or assembly. Using the object modules generated by the language translators (Assembler, COBOL, or FORTRAN compilers), the Linkage Editor routine prepares loadable programs (also called load modules) which are in a format suitable for execution. The Linkage Editor routine may be executed either conversationally or nonconversationally.

Processing Utilities

VMOS contains two Processing Utilities: the Message Processing routine and the Sort/Merge routine. The Message Processing facility operates under direct control of the Control Program and is summarized in the Control Program portion of this section. The Sort/Merge routine is a generalized system which runs as either a foreground or background task. Its primary objective is the efficient processing of files stored on direct access devices. Its secondary objective is the processing of magnetic tape files for conventional batch processing applications.

Display Utilities

VMOS supplies its users with two display utilities: the Spoolout routine and the DPAGE routine. The Spoolout routine operates under direct control of the Control Program and is summarized under Job Control in this section.

DPAGE is a general-purpose utility routine that provides the VMOS user with the facility to display or print files or volumes, and to manipulate data in a page or in the 16-byte key. It may be used as a conversational or nonconversational task. Display output data is dispatched to the the user's terminal in conversational mode, or to the printer in nonconversational mode. The print facility is normally for large-volume output and the output is always sent to the printer.

Peripheral to Peripheral Utilities

VMOS provides utility programs that will transcribe data directly from one peripheral device to another. There are ten such programs.

The Card-to-Tape routine (CDTP) transcribes 80-column card records or paper tape to magnetic tape in standard Series 70 format. The output tape file contains standard Series 70 labels and may be single or multivolume.

The Selective Card-to-Printer and/or Punch routine (CDPR) transcribes 80-column card records or paper tape to punched cards or paper tape and/or to the printer. Card files are punched in EBCDIC and the final card contains /* in the first two columns to signify the end of the file. Printed output may be in character mode (EBCDIC graphics) or hexadecimal mode (two digits per character); the print format may be List or Display.

The Card-to-Random Access routine (CDRA) transcribes 80-column card records or paper tape to a random access file. Input cards are punched in EBCDIC format, with the final card containing /* in the first two columns to signify the end of the file.

The Selective Tape-to-Printer and/or Punch routine (TPPR) transcribes data from magnetic tape to punched card and/or to the Printer. The input volume may be labeled or unlabeled. If labels are used, a multivolume input file may be processed. Card output files are punched in EBCDIC with the final card containing /* in the first two columns to signify the end of file. Printed output may be in character mode (EBCDIC graphics) or hexadecimal mode (two digits per character); the print format may be List or Display.

The Tape-to-Tape routine (TPTP) transcribes data from one magnetic tape to another. The input and output tape blocks can range in size from 12 to 4096 characters, and can contain fixed-length records, variable-length records, or records of undefined size. Except for records of undefined size, records can be blocked or unblocked. In the case of fixed-length records, input fields can be field-selected, packed, or unpacked during the copying process. Tape volumes may be labeled or unlabeled, single or multivolume.

The Tape-to-Random Access routine (TPRA) transcribes data from magnetic tape to a random access file. Input tape blocks can range in size from 12 to 4096 characters and may contain fixed-length records, variable-length records, or records of undefined size. Except for records of undefined size, records may be blocked or unblocked.

The Random Access-to-Printer and/or Punch (RAPR) routine transcribes data from a random access file to punched cards or paper tape and/or to the printer. Card output files are punched in EBCDIC with the final card containing /* in the first two columns to signify the end of file. Printed output may be in character mode (EBCDIC graphics) or hexadecimal mode (two digits per character); the print format may be List or Display.

The Random Access-to-Tape routine (RATP) transcribes data from a random access file to a magnetic tape. Output tape blocks can range in size from 12 to 4096 characters, and may contain fixed-length records, variable-length records, or records of undefined size. In the case of fixed-length records, records may be blocked or unblocked. Tape volumes may be labeled or unlabeled, single or multivolume.

The Tape Edit routine (TPEDIT) displays all, or selected portions, of a magnetic tape on the on-line printer. The contents of the tape may be displayed in Character mode (EBCDIC graphics), hexadecimal mode (two digits per character), or both. At end-of-job, the programmer can terminate the routine or enter additional parameters. In this way, different portions of the same tape (or another tape) can be edited.

The Tape Compare Routine (TPCOMP) is a diagnostic aid used to compare information recorded on one magnetic tape with that of a second magnetic tape. Output from this routine consists of a listing of all portions of the two tapes that did not match.

Language Processors

VMOS provides the programmer with a full range of language processors: an assembler, standard COBOL and ANSI COBOL compilers, Report Program Generator compiler, and a FORTRAN compiler. The detailed description of the use of these processors is contained in the VMOS Programmer's Reference Manual.

COBOL Background Compiler

The VMOS COBOL Background Compiler accepts programs written in the same COBOL language available on other Series 70 systems. Except for minor source language differences, the COBOL Background Compiler is functionally identical to the ANSI COBOL compiler.

ANSI COBOL Background Compiler

The ANSI COBOL Background Compiler accepts COBOL source programs written in the USA Standard COBOL. The ANSI COBOL compiler is an enhancement to the Background COBOL compiler. The compiler provides the user with the following capabilities:

Source input can be retrieved from a cataloged file on disc, from a card deck, from a remote terminal, or from a disc-resident COBOL source-library file.

The compiler generates either Class I or Class II programs, depending upon a user-supplied parameter.

The generated object module is written on disc.

The compiler is disc-oriented in that it uses temporary EAM disc files for intermediate work space.

Requested listings can be written on temporary files for automatic printing at task termination, or on cataloged files so that the user may optionally request printing after task termination.

Report Program Generator Compiler

The VMOS Report Program Generator (RPG) Compiler accepts programs written in the same RPG language available on other Series 70 systems. It is used to produce a printed report without requiring a detailed knowledge of machine coding. Common report features such as input data selection, editing, calculating, summarizing, control breaks, and file updating are provided.

FORTRAN IV Background Compiler

The VMOS FORTRAN IV Background Compiler accepts programs written in the same FORTRAN language available on other Series 70 systems. The compiler generates highly optimized object coding for use as both production programs and subprograms.

VMOS Assembler

The VMOS Assembler provides a machine-oriented, symbolic programming language for use by machine language programmers. This symbolic programming language supports three general types of programming statements.

Assembly instruction statements are a one-for-one symbolic representation of actual machine instructions. The Assembler produces an equivalent machine instruction in the object program for each assembly instruction statement in the source program.

Assembly control statements provide auxiliary functions that assist the programmer in checking and documenting his programs, in controlling the assignment of storage addresses, in program sectioning and linking, in defining data and storage fields, and in controlling the Assembly System itself. Assembly control statements specify the auxiliary functions to be performed by the assembler, and, with a few exceptions, do not result in the generation of any machine language code during the assembly.

VMOS Interactive Routines

VMOS provides its users with a comprehensive set of interactive routines for program preparation, using FORTRAN and COBOL, BASIC language programming and file editing. The routines are described in full in a set of eight VMOS Interactive Languages Reference Manuals.

Conversational FAST FORTRAN

FAST FORTRAN was designed to produce very fast problem solutions. Programs to solve such problems are generally of the one-shot variety. FAST FORTRAN provides (1) very fast compilation time, (2) good diagnostic capability, (3) minimum system overhead, and (4) gives the nonprofessional programmer who needs an immediate answer an easy-to-use, fast, conversational system that accepts FORTRAN input.

COBOL Program Development Subsystem (CODE)

The COBOL Program Development Subsystem (CODE) is a facility that supervises and assists a user in preparing, editing, debugging, compiling, and executing a COBOL program. To facilitate these actions, CODE accepts free-form input, allows the user to define shorthand abbreviations, offers text editing, has simplified compilation and execution commands, and maintains a file of compilation diagnostic messages that may be interrogated during editing.

BASIC

VMOS BASIC provides a fast, complete program preparation and execution facility for the easy-to-use BASIC language. BASIC provides comprehensive editing facilities that permit the user to create, modify, list, execute, save, and reload his source program. BASIC also incudes extensive syntax checking for both single statement and global type errors.

VMOS Editor (EDT)

The VMOS Editor (EDT) is an interactive text editor that allows a user to create, delete, copy, compare, and concatenate files and to add, delete, and modify text within a file. It is a short, fast, easy-to-use editor containing those facilities that are most commonly used and most often needed. It may be executed either conversationally or nonconversationally, and may be either sharable or nonsharable.

File Editor

The File Editor may be used either conversationally or nonconversationally in order to create, edit, and display files. The basic File Editor commands carry a rich and detailed snytax. This richness offers the user the ability to adapt each command to a great variety of different editing operations. Thus, a typical editing command may operate on a single line, a part of a line, on a single character, or upon particular syllables in a whole succession of lines.

GENERAL

This section discusses certain features of the Control Program, with which a programmer operating in a system management environment should be familiar. These features are job control and spoolout. Although the programmer can exert limited control over their operation, a knowledge of the way in which they function will enable the programmer to better manage system throughput. In addition, VMOS memory organization and the function of stacks are also described.

JOB CONTROL

This Control/Program component manages those system functions which are (with some exceptions as will be noted) nonconversational in nature. The Job Controller monitors nonconversational tasks. System Accounting maintains disc based accounting information, and System File Management establishes the environment for processing "SYS" type files used by the Job Controller, Spoolout, and Accounting.

The Job Controller is a collection of routines that govern the actual processing of nonconversational jobs. These routines have exclusive control over all nonconversational system input/output and scheduling functions. The Job Controller allocates system resources, schedules work and provides for the disposition of system output. In addition, it provides for the optimum scheduling of batch jobs in the system based on priority, estimated CPU time, and device requirements.

The system accounting routine maintains a file of task accounting information that can be written to tape at system shutdown time. The records are written in sequence according to task termination time and not by task sequence number (TSN).

System file management is closely related to spoolout and has as its main purpose the most efficient use of the task assigned system output devices.

SPOOLOUT

Spoolout is the Control Program routine which performs the printing and punching of files under VMOS. The routine can be activated by the user (see VMOS Service Routines Manual), and is automatically activated, when necessary, by the system when a task is terminated.

Spoolout jobs are maintained on Spoolout queues. If there are jobs waiting to be spooled out and if the device is available, the first job in the queue is activated; and any PRINT or PUNCH jobs or new terminating jobs requiring Spoolout are placed on the queue. When a Spoolout task is completed, any temporary direct access space occupied by the output file is restored to the system. Output file examples would be assembly listings, object decks, and system output messages. A Spoolout job not completed during a VMOS session may be recovered in the next.

STACKS

Associated with each task are three or more storage areas called stacks. Stacks normally reside in Class 3 memory (core resident, nonpagable) and their primary purpose is to provide an area for register storage when a task is interrupted. The system dynamically generates stacks for each user-written program and for each active system routine associated with that task.

A task running in processor state-1 (P1), initially has two stacks: a P2 stack and a P1 stack. These stacks are allocated at task initiation time, and the P1 stack is attached to the task at execution. These two stacks exist for the task's duration. The P2 stack is called the permanent P2 stack.

A special type task, called Interrupt Driven, is allocated a P2 stack only. Such tasks (Spoolin and Spoolout, for example) do not operate in processor state P1. Another task type, called Permanent System, also has only a P2 stack.

VIRTUAL MEMORY ORGANIZATION

To regulate and simplify its control by VMOS memory management, a virtual memory is divided into six classes as depicted in figure 1-1.

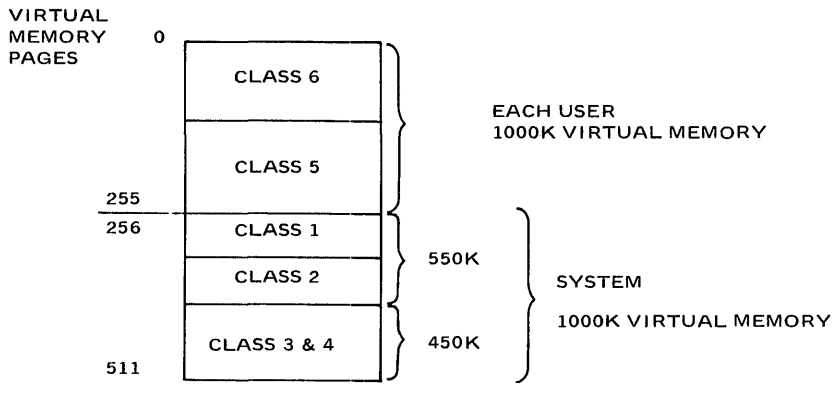


FIGURE 1-1. VIRTUAL MEMORY CLASS DIVISION

The six virtual memory classes and their contents are:

Class 1 — the resident control program, privileged and nonpagable.

Class 2 — which is occupied by the nonresident portions of the control program, privileged, and pagable.

Class 3 — is occupied by dynamically acquired resident portions of the control system, privileged, and nonpagable. This memory is used for task control blocks, terminal I/O buffers, and certain system work space.

Class 4 — is occupied by the nonresident work space dynamically acquired by the control program and by shared code called by the users of the system. All Class 4 pages are pagable and have drum images. The pages utilized by the control program are privileged but those utilized by the user are not.

Classes 1 through 4 constitute the system's virtual memory and are contained within the one-million bytes of address space available to the system.

Class 5 — is occupied by dynamically allocated pagable areas acquired for the specific user by the control program. This class is used for task-dependent virtual memory tables, protected file control blocks, program loader data, data maintained by the interactive debugging language, and I/O buffers acquired for the task by the system.

Class 6 — is occupied by dynamically allocated pages acquired by the user for code and work areas. These are under control of the user task.

Classes 5 and 6 constitute the users virtual memory and are limited to one-million bytes of address space per user.

INTERCOMPUTER COMMUNICATION

VMOS facilitates computer-to-computer communication through support of the Data Exchange Control (DXC). The DXC is a hardware device through which messages can be channeled between two processors. The joining of a real memory operating system (RMOS) with VMOS, in an auxiliary capacity, greatly increases the number of interactive terminals available to VMOS as input devices.

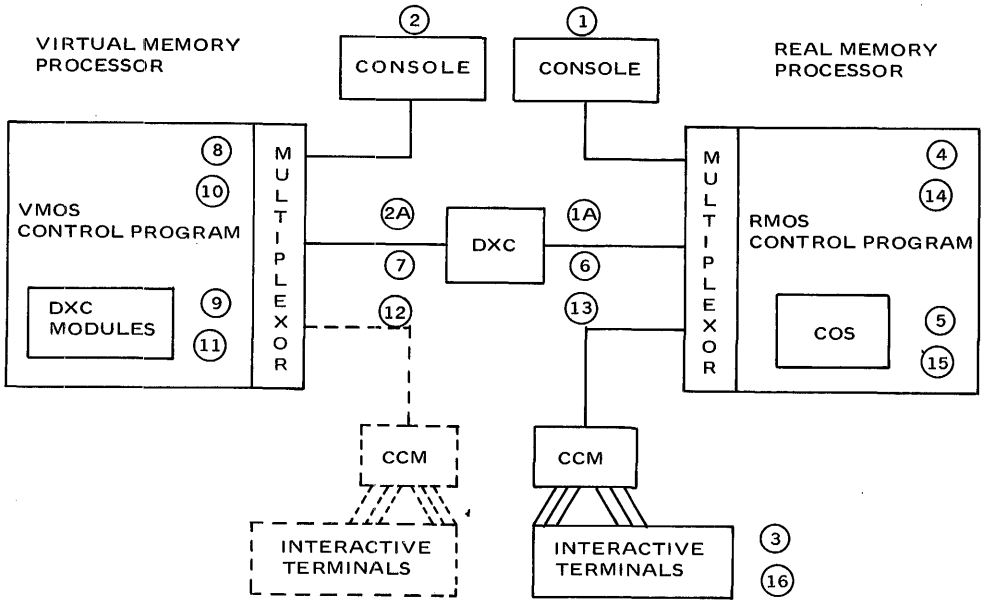
VMOS exchanges messages with the auxiliary real memory system on an inquiry response basis. VMOS reads messages initiated by a terminal based user, processed by the auxiliary computer and, upon completion of its reading function, transmits the appropriate response to user via the auxiliary processor. This process is transparent where processing requests are initiated by the user, who is communicated with by the system as if the operations were in standard VMOS interactive environment.

In the VMOS processor, all message receptions (reads) and transmissions (writes) are consolidated and distributed by the Control Program. There are two categories of reads and writes: one for the Control Program (privileged) and the other for user-written programs and software-supplied routines. VMOS receives reads (from RMOS as interrupts) which the Control Program analyzes and passes to supporting DXC software. The DXC software verifies the transmission, decodes it, and transfers the request back to the Control Program for processing by VMOS software. When processing is complete, the Control Program-DXC software interaction is renewed resulting in the transmission of a response to RMOS.

The operation of a real memory system in the processing of transmissions to and from the DXC is analogous to complementary operations in VMOS. The only difference relates to the processing of messages received by the Control Program. In the real memory processor, messages are verified, decoded, and passed between the user and the Control Program by Communications Oriented Software (COS). Thus, any real memory system used in an auxiliary capacity with VMOS in conjunction with the DXC must contain COS as part of its operating system. The portion of COS which performs the bulk of message processing is the Communication User Program (CUP). It is the responsibility of real memory system programming personnel to prepare an appropriate CUP. A general description of CUP function and use may be found in the COS Functional Description Manual.

The conjunction of VMOS and RMOS via the DXC does not restrict the normal operation of either system. Real memory batch processing and other COS operations may be undertaken without regard to the resources used for VMOS-RMOS message processing. Similarily, VMOS will continue to process local and remote batch task, tasks entered from terminals attached directly to the system, and CAM programs. However, CAM programs cannot be activated by users communicating with VMOS from the auxiliary processor. Also, the output generated by PRINT and PUNCH commands will occur at the VMOS facility, not the RMOS facility.

Figure 1-2 diagrams the message flow between a Virtual Memory System and a Real Memory System across a DXC and table 1-1 lists the message flow operations.



NOTE: REFER TO TABLE 1-1 FOR REFERENCED MESSAGE FLOW OPERATIONS

FIGURE 1-2. VMOS-DXC-RMOS MESSAGE FLOW

TABLE 1-1. VMOS-DXC-RMOS MESSAGE FLOW OPERATIONS

VMOS Operation	DXC Operation	RMOS Operation
(2) Operator loads virtual memory and activates DXC interface (/START DXC, X'da' command). (2A) DXC portion of Control Program transmits initialization message to DXC.		(1) Operator loads real memory operating system including communication oriented software (COS). (1A) COS notifies DXC that it is ready to read initialization message from VMOS. (3) User initiates request for VMOS processing. (4) Control program analyzes resulting interrupt and passes control to COS. (5) COS processes transmission from terminal and transfers it to DXC.
	(6) DXC received the transmission and checks to ensure that VMOS is prepared to accept the transmission.	
(7) VMOS accepts transmission. (8) Control program analyzes resulting interrupt and passes control to DXC software.		

(Continued)

TABLE 1-1. VMOS-DXC-RMOS MESSAGE FLOW OPERATIONS (Continued)

VMOS Operation	DXC Operation	RMOS Operation
(9) DXC software analyzes transmission and passes processing request portion to Control Program. (10) Control Program initiates required processing, and upon completion of processing passes control to DXC software. (11) DXC software formats processing information for transmission to RMOS, and passes transmission to DXC.		
	(12) DXC receives the transmission and checks to ensure that RMOS is prepared to accept the transmission.	
		(13) RMOS accepts transmission. (14) Control program analyzes resulting interrupt and passes control to COS. (15) COS processes transmission and passes it to user's terminal. (16) User receives processing information in response to his request (3).

Note: See figure 1-2 for message flow chart.

Part 2
Control Program Usage Concepts

GENERAL

The VMOS Control Program, as described in Section 1, provides comprehensive facilities for managing the system's processing environment. These facilities relate to the functions of Task, File, and Device Management, and are accessible to the programmer through the system's Command Language.

VMOS also supplies the user with a basic communications capability, the Communications Access Method (CAM), and the facility for implementing computer-to-computer communication via DXC (Data Exchange Control – see Section 1). The programmer also manages these functions through facilities of the control program.

The sections constituting this part describe the facilities available under VMOS for Task, File, and Device Management, and the implementation of CAM and DXC. The initial section discusses the basic concepts relative to Tasks, Files, and Devices, and to the function of the VMOS Command Language. It also contains four decision-oriented tables relating various directives made available by the Command Language to the process of managing work through the system. Table 2-1 provides a guide to the use of the three decision tables based on the function the programmer wishes to perform, e.g., task or program initiation, file maintenance, device disposition, etc.

TABLE 2-1. GUIDE TO SYSTEM MANAGEMENT DIRECTIVE TABLES

Programming Function	Process Initiation (Table 2-3)	Processing Control (Table 2-4)	Process Termination (Table 2-5)
ACCESS METHOD MANAGEMENT		X	
DEVICE (SPACE) ALLOCATION	X		
DEVICE (SPACE) DEALLOCATION			X
DEVICE (SPACE) REGULATION		X	
FILE CREATION	X		
FILE DISPOSITION			X
FILE MAINTENANCE		X	
PROGRAM DIRECTION		X	
PROGRAM INITIATION	X		
PROGRAM-TO-PROGRAM COMMUNICATION		X	
PROGRAM TERMINATION			X
REMOTE JOB INITIATION	X		
REMOTE JOB REGULATION		X	
REMOTE JOB TERMINATION			X
TASK INITIATION	X		
TASK REGULATION		X	
TASK INTERRUPTION			X
TASK TERMINATION			X

GENERAL

The performance of system management functions requires a foreknowledge of certain basic concepts. These are the function of the VMOS Command Language; the way in which VMOS perceives Tasks, Files, and Devices; the concept of volumes; and the relationship among these three elements.

COMMAND LANGUAGE

The principal means of communication between the user and control program is the system's Command Language. This language comprises a set of commands and macros (directions) designed to facilitate the management of system facilities.

The commands contained within the Command Language constitute the primary facility for direct communication with the system for the performance of work. These directives are the means by which the programmer causes the execution of both system routines and user problem programs.

Command Language commands can be used in the conversational (interactive) or nonconversational (background) mode. When operating in the conversational mode, the programmer uses Command Language commands to maintain a dialogue with the system while it performs desired work. When used in the nonconversational mode, the commands are analogous to a Job Control language because the instructions are submitted in prepared form and are not monitored by the programmer.

Macro directives supplied by the Command Language enable a programmer to incorporate features of the Command Language into Assembly language programs. This permits the programmer to establish communication between a problem program and the system.

TASKS

Within context of VMOS processing, the term task denotes definite relationship between the programmer and the system. Basically, this relationship defines a task as encompassing the work undertaken by the system in response to Command Language directives issued by the programmer. VMOS differentiates this work into primary and secondary tasks. The system makes reference to each task by a 4-digit number called a task sequence number (TSN).

Primary Task

The term primary task identifies the extent of a programming session, that is, the span of activity beginning with system's recognition of the programmer and terminating when the programmer severs the connection. The system recognizes a programmer when it receives his LOGON command. The programmer severs the connection by issuing a LOGOFF command. When reduced to simplest terms, a primary task is the work performed by the system during the period between a programmer's LOGON and LOGOFF commands.

LOGOFF Processing

When the user terminates a programming session (logs off), the system initiates the procedures necessary to return resources (required by the processing demands of the primary task) to the system. This is known as LOGOFF processing, the logic of which provides the system with a list of any files created by the system to handle the program's output. If these files are printed, they are considered to be part of the programmer's primary task, and associated with it by the primary task number. This is the task sequence number (TSN) assigned to the session when it was initiated. If the file is to be punched into cards, the punching process is regarded by the system as a new task, and as such is identified by a new task number.

Secondary Task

Secondary task is the term applied to any processing initiated within a programming session that is identified by a task number other than the TSN associated with the primary task. The punching of a file onto cards as described above (under LOGOFF processing) is an example of a secondary task.

Two other processing conditions generate secondary task. The first is the use of Command Language directions by a user program or by a system routine, that call for the creation of printed or punched output. The second is the execution of an ENTER file (refer to Procedure Files discussion).

All secondary tasks created as the result of print or punch operations carry the same priority as the primary task in which these operations were initiated. However, the secondary task associated with execution of an ENTER file may carry lower priority than the initiation task because ENTER files are always processed in the background (nonconversationally).

PROGRAM CLASSIFICATION

VMOS differentiates between programs that make use of virtual memory (pagable programs) and programs that reside wholly in main memory (nonpagable programs). Pagable programs are referred to as Class I programs; nonpagable as Class II programs.

Class I Programs

Class I programs are restricted to the physical memory addressing capacity of the system. These programs must reside in physical memory throughout their execution and require that physical memory be assigned to them on a contiguous basis.

Class II Programs

Class II programs do not require contiguous main memory for their execution and may reside in the system's virtual memory. These programs are broken into pages (4096 bytes), use the system's Virtual Address Mode, and are normally passed on a page basis between the system drum and main memory.

FILES

In VMOS, a file is the principal processing medium and is defined as any named collection of related records. All information entering, leaving, and contained within the system resides in a file. The term file, therefore, not only refers to collections of source program statements, object program code, and data created by the programmer but also encompasses the routines supplied for the programmer's use by the system. In addition, VMOS has designated files set of input/output media common to all users. This latter category of files are the System Files and the System's Task Library.

System Files

The term system files defines a set of system-supported input/output streams supplied in common for all users by VMOS. At task initiation time, these files are automatically created for, and associated with, each task. When a task is terminated, the system files created for it are erased.

The names of system files are as follows:

SYSCMD

SYSDTA

SYSIPT

SYSLST

SYSOPT

SYSOUT

The first three listed above are input files, and the last three are output files. All six reside on direct access devices for duration of the task for which they were created.

SYSCMD

The SYSCMD file is supplied by the system to contain programmer's control language commands, and is read for him by the System Command Language Processor. For tasks operating in the interactive mode, VMOS assigns SYSCMD to the user's terminal. For background mode tasks, SYSCMD is read from the spooling stream associated with the card reader, a remote batch terminal, or a cataloged procedure file. A programmer can neither change the assignment of SYSCMD nor have his problem program read from it.

SYSDTA

SYSDTA is the file created by the system to contain data (as opposed to Control Language commands) being read into the system. Most system components read their input from this file, and a programmer may designate that his problem program read its input from SYSDTA. At task initiation, SYSDTA is assigned to the same sources as those described for SYSCMD. However, a programmer may direct SYSDTA away from its primary assignment to a direct-access-resident, to a cataloged file or to the card reader. Subsequently, the programmer may direct SYSDTA back to its primary source. The discussion of the SYSFILE command (Part 3, Section 2) describes the manipulation of SYSDTA.

SYSIPT

The SYSIPT file, similar to the SYSDTA file, is supported to supply compatibility with the Tape and Tape/Disc Operating Systems (TOS and TDOS). For background tasks, the primary assignment of SYSIPT is the same as that for SYSDTA. However, for conversational tasks, the system does not create a SYSIPT file at task initiation. Neveretheless a programmer can define a SYSIPT file while operating in conversational or background modes (see SYSFILE command). Whether it is defined by the programmmer or assigned by the system, the programmer can direct SYSIPT to the same secondary sources as SYSDTA.

SYSLST

The SYSLST file is the system-created vehicle for printer-destined information. VMOS system-supplied programs automatically use this file to handle substantial output such as program listing, dumps, etc. Programmers should use it similarly for the output of their problem programs. The system accumulates records in the SYSLST file until user logs off, and then sends the entire file to the printer. After being sent to the printer, SYSLST is erased. SYSLST cannot be directed to a secondary source by the programmer.

SYSOUT

The SYSOUT file is maintained by the system so that output of generally smaller quantity can be distinguished from large volume SYSLST output. VMOS uses SYSOUT as a vehicle for diagnostic messages, response messages, etc., and programmers should use it similarly for small quantity output from their problem programs. When a task is operating in the interactive mode, the system directs SYSOUT to the user's terminal. For background tasks, the system implements SYSOUT in the same manner as SYSLST. SYSOUT cannot be directed to a different output device by the programmer.

SYSOPT

The SYSOPT file is supported primarily to supply compatibility with the Tape and Tape/Disc Operating Systems (TOS and TDOS) for the creation of punched output. When records are directed to SYSOPT, by system components or by user's problem program, the system collects them in a temporary disc file, and has them punched on cards at a task termination. The user cannot direct SYSOPT to a different output device.

System Task Library (TASKLIB)

The System Task Library (TASKLIB) is a file containing most of the VMOS-provided software products that are not in load module form. The TASKLIB is, then, the system's object module library (OML). This file is used by the Dynamic Linking Loader (DLL) and Linkage Editor to satisfy unresolved external references. In other words, these routines search the TASKLIB directory in an attempt to locate entry points or csect-names which match those of the unresolved EXTRNS. Search methods employed by the Linkage Editor and DLL are described in the VMOS Service Routines Manual.

DEVICES

VMOS recognizes two basic classifications of devices: public and private. The nature of a device, or the data medium it supports, determines whether or not a device is public or private. Public devices provide for concurrent control by more than one task. Conversely, private devices are restricted to the control of one task at a time.

Public Devices

In VMOS, a public device must be a direct (random) access device. The following list shows the general classification of public (direct access) devices supported by VMOS:

Disc Drives,

Mass Storage Unit (MSU),

Drum Unit, and

Control Units with multichannel switch features.

These direct-access devices, although generally considered public devices, may be designated private by the user. VMOS does not restrict them to public classification.

Private Devices

Unit Record (UR) devices and Sequential Access (SA) devices constitute VMOS private devices. The following UR devices are supported by VMOS:

Card Readers,

Printers (132 and 160 column),

Bill Feed Printer,

Card Punches, and

Paper Tape Reader/Punch.

The remaining private devices supported by VMOS are the following SA devices:

Magnetic Tape Units (7- and 9-level tapes),

Magnetic Tape Units (9-level phase encoding), and

Control Units configurations with 7-level tape features.

VOLUMES

VMOS uses the concept of a volume as the bridge between the file and the device used to access it. A volume is the data medium portion of a device configuration: the medium within which files reside. The term volume refers, therefore, to disc packs, MSU magazines, and tape reels. Because the data storage portion of a drum cannot be separated from the remainder of the device, drums may be considered as either volumes or devices.

As it is impractical to use UR devices as data storage media, and because these devices cannot be directly referenced during execution of Class II programs, VMOS does not associate the concept of volumes with them. Although UR devices may be directly accessed by Class I programs, frequent use in this manner is discouraged since it leads to inefficient use of the system.

File/Volume Relationship

A file is said to be stored in the system if it resides on one or more direct-access or magnetic tape volumes, and if the identification of these volumes (volume serial number) is available in the system catalog. In VMOS, a volume may be a removable disk pack, a reel of magazine tape, or a mass storage unit magazine. For direct-access volumes, more than one file may be contained on a volume. For magnetic tape files, however, only one file can be contained on a volume (that is, multifile reels are not allowed); however, a tape file can be on more than one volume.

Public and Private Volumes

As in the case of devices, a volume may be classified as public or private. A public volume is a direct-access volume and must be mounted and be on-line during the entire period of system operation. A public volume may be used for many tasks concurrently. A private volume need not be mounted during the entire period of system operation. Its use is restricted to one task at a time and it needs to be mounted only when the task refers to it. A direct-access volume may be a private volume, while magnetic tape volumes are always classified as private volumes.

> Note: All volumes of a direct-access file must be mounted in order to access the file.

Use of Public and Private Volumes

Files to be cataloged can be stored on public or private volumes. The system assumes that a file is to be stored on a public volume unless the programmer specifically asks for its storage on a private volume. Programmers generally make the most effective use of the system by storing their files on public volumes. Since public volumes are always on-line, files stored on them are always available for access to a user's task. The user can allocate space for this files on public volumes within the limits of public space allocation established for him by the system controller. The programmer may also specify that a file be contained on specific public volumes (refer to FILE command, VOLUME=parameter – Part 3, Section 2). However, if volumes specified by the programmer do not contain sufficient space for the file, the program will be terminated.

If a programmer employs private or TOS volumes, he may need to wait for a device on which to mount the volume since the system must determine whether or not it can honor the request. Note that the SECURE command should be used to obtain all necessary private volume devices. The securing of devices is described in Section 4, under Device and Spare Management.

File Space on Public Volumes

If the programmer's public space allotment is exhausted in the process of inserting or adding records to a file, the system transfers control to the exit address of the problem program. The program can then take appropriate action: for example, make space available by deleting an old file.

System workfiles created on behalf of a programmer by software-supplied programs (for example, workfiles created when executing the VMOS Assembler) are stored on public volumes, but the space is not charged to the programmer's public space allotment, since the Evanescent Access Method (EAM) is typically employed. EAM description is given in Section 3; however, it is pertinent to note here that EAM provides for processing of temporary direct access files.

File Space Allocation

The programmer may specify the primary space requirements for files on direct access volumes, and may also specify the amount of secondary space to be allocated when the primary area is full. This can be accomplished by the FILE command (see Part 3, Section 2). Additionally, this specification may be defaulted, in which case the installation standard space allocation established at system generation time is used. (Check with the System Controller for this default value.)

The system also obtains additional space dynamically (that is, while the file is opened) whenever this is required. For files which reside on public volumes, the system first attempts to obtain more space on the last volume which contains the file. If no space is available on that volume, the system chooses the public volume which has the most available space. However, if space exhaustion occurs for files that reside on private volumes, the system transfers control to an address specified in the exit parameter of the problem program.

TOS Volume

VMOS supports a third type of volume: the TOS volume. A TOS volume need not be mounted during the entire period of system operation. Its use is restricted to one task at a time and it need only be mounted when the task refers to it. Since TOS I/O processing is allowed, file protection is not guaranteed.

VMOS access to TOS tape volumes is provided through the sequential access method (SAM) and the basic tape access method (BTAM) provided that physical tape blocks do not exceed 4096 bytes. VMOS access to TOS direct-access volumes is not provided. Note that Class I programs, however, can freely access TOS direct-access volumes. In other words, Class I program compatibility is fully maintained.

System Volume

The VMOS System Volume is a VMOS private direct-access volume that may be shared and is so defined by the operator's SETUP command (see VMOS Operations Management Reference Manual).

Volume Characteristics

The salient characteristics of VMOS volumes are outlined in table 2-2. Table 2-3 lists the Command Language commands and macros relating to process initiation; table 2-4 to processing control; and table 2-5 for process termination.

TABLE 2-2. SUMMARY OF VOLUME CHARACTERISTICS

Volume Characteristics	Public	Private	TOS
Storage media	Direct-access devices	Direct-access devices	Direct-access devices
		Magnetic tape	Magnetic tape
Number of tasks which may concurrently use the volume.	Any number	1 Any number if volume resides on a system direct access device.	1
File Protection	Full VMOS file protection (see File Security in Section 3).		None.
Processing methods allowed	VMOS DMS access methods. VMOS I/O macros which process system files. TOS monitor macros which process TOS system files.		TOS FCP and physical level I/O.
Label support	Standard file labels but no user labels.	For direct-access devices, standard labels; no user labels supported. For magnetic tape, labels may be standard or omitted. Additionally, user header and trailer labels are supported on magnetic tape.	Same as TOS.
Volume serial number	6-character alphanumeric field; first 3 characters must be PUB.	6-character or less alphanumeric field; first 3 characters must not be PUB.	
Volume is mounted.	At start of time-sharing session.	When requested or required by a task.	
Volume is dismounted.	Only at termination of time-sharing session (i.e., never at task termination).	When requested or required by the task. If a system volume, dismount is effected by the SETUP command.	When disconnect is requested or required by a task.
Space dynamically acquired while executing the problem program.	Yes	Yes	No
SECURE command required.	No	Yes, unless user wishes to risk being aborted if no I/O device is available. System volumes require SETUP command.	

TABLE 2-3. SYSTEM MANAGEMENT PROCESS INITIATION DIRECTIVES

Applicable Command/Macro	Task Initiation	Remote Job Initiation	Program Initiation	File Creation	Device (Space) Allocation
CATALOG Command				FM	
CATAL Macro				FM	
DATA Command				FM	
DO Command			TM		
ENTER Command	TM		TM		
EXECUTE Command			TM		
FCB Macro				FM	
FILE Command/Macro				FM	DM
IDFCB Macro				FM	
LOAD Command			TM		
LOGON Command	TM				
OPEN Macro				FM	
PASSWORD Command				FM	
PROCEDURE Statement			TM		
REQM Macro					DM
RLOGON Command		TM			
RSTART Command					DM
SECURE Command					DM

LEGEND

DM — DEVICE MANAGEMENT
[Part 3, Section 3-
Command/Macro Descriptions]

FM — FILE MANAGEMENT
[Part 3, Section 2
Command/Macro Descriptions]

TM — TASK MANAGEMENT
[Part 3, Section 1
Command/Macro Descriptions]

TABLE 2-4. SYSTEM MANAGEMENT PROCESSING CONTROL DIRECTIVES

Applicable Command/Macro	Task Regulation	Remote Job Regulation	Program Direction	File Maintenance	Device (Space) Regulation	Interprogram Communication	Access Method Definition	File Reconstruction
BTAM Macro							FM	
CALL Macro			TM			TM		
CATALOG Command				FM				
CATAL Macro				FM				
CHANGE Command				FM	DM	FM		
CHNGE Macro				FM	DM	FM		
CHKPT Macro			TM					
CLOSE Macro				FM			FM	
COPY Command/Macro				FM				
CSTAT Macro			TM					
DROP Command				FM	DM	FM		
EAM Macro							FM	
ELIM Macro							FM	
EOF Command			TM					
EXCOM Macro			TM			TM		
EXECM Macro			TM			TM		
FEOV Macro							FM	
FSTATUS Command				FM				
FSTAT Macro				FM				

(Continued)

TABLE 2-4. SYSTEM MANAGEMENT PROCESSING CONTROL DIRECTIVES (Continued)

Applicable Command/Macro	Task Regulation	Remote Job Regulation	Program Direction	File Maintenance	Device (Space) Regulation	Interprogram Communication	Access Method Definition	File Reconstruction
GET Macro							FM	
GETFL Macro							FM	
GETKY Macro							FM	
GETR Macro							FM	
GETSW Macro			TM			TM		
HOLD Command				FM	DM	FM		
INSRT Macro							FM	
LOADM Macro			TM			TM		
LOG Macro								FM
LPOV Macro			TM			TM		
OPEN Macro				FM			FM	
PAM Macro							FM	
PARAMETER Command			TM					
PASS Macro			TM					
PROUT Macro			TM					
PUT Macro							FM	
PUTX Macro							FM	
RDATA Macro			TM					
RDCRD Macro			TM					

(Continued)

TABLE 2-4. SYSTEM MANAGEMENT PROCESSING CONTROL DIRECTIVES (Continued)

Applicable Command/Macro	Task Regulation	Remote Job Regulation	Program Direction	File Maintenance	Device (Space) Regulation	Interprogram Communication	Access Method Definition	File Reconstruction
RECON Command/Macro								FM
RELEASE Command				FM		FM		
REL Macro				FM				
RELSE Macro							FM	
RESET Command								FM
RESTART Command			TM					
RESUME Command	TM							
RETRN Macro			TM			TM		
RJOB Command		TM						
RMSG Command		TM						
ROUT Command		TM						
RSTATUS Command		TM						
SAVE Macro						TM		
SETL Macro							FM	
SETSW Command/Macro	TM		TM			TM		
SKIP Command	TM		TM			TM		
STATUS Command			TM					
STEP Command	TM					TM		
STORE Macro							FM	

(Continued)

TABLE 2-4. SYSTEM MANAGEMENT PROCESSING CONTROL DIRECTIVES (Continued)

Applicable Command/Macro	Task Regulation	Remote Job Regulation	Program Direction	File Maintenance	Device (Space) Regulation	Interprogram Communication	Access Method Definition	File Reconstruction
SYSFILE Command					DM			
TMODE Macro	TM							
TOCOM Macro			TM			TM		
YPASS Macro			TM					
WRLST Macro			TM					
WROUT Macro			TM					
WRTOT Macro			TM					
WRTRD Macro			TM					

LEGEND

DM — DEVICE MANAGEMENT
[Part 3, Section 3
Command/Macro Descriptions]

FM — FILE MANAGEMENT
[Part 3, Section 2
Command/Macro Descriptions]

TM — TASK MANAGEMENT
[Part 3, Section 1
Command/Macro Descriptions]

TABLE 2-5. SYSTEM MANAGEMENT PROCESS TERMINATION DIRECTIVES

Applicable Command/Macro	Task Termination	Remote Job Termination	Program Termination	File Disposition	Device (Space) Deallocation	Process Interruption
ABEND Command	TM					
BREAK Command						TM
CANCEL Command	TM					
CLOSE Macro				FM		
ENDP Statement			TM			
ERASE Command/Macro				FM		
EXIT Macro						TM
EXITP Macro						TM
EXLST Macro				FM		
FILE Command/Macro				FM	DM	
INTR Command						TM
LOGOFF Command	TM					
PAUSE Command						TM
RELEASE Command				FM	DM	
REL Macro					DM	
RELM Macro					DM	
RESUME Command						TM
RLOGOFF Command		TM				
RSTOP Command					DM	
SETIC Macro						TM
SPEXT Macro						TM
STXIT Macro						TM
SYSFILE Command				FM		
TERM Macro			TM			
TERMD Macro			TM			
TERMJ Macro			TM			

LEGEND

DM — DEVICE MANAGEMENT [Part 3, Section 3 Command Macro Descriptions]

FM — FILE MANAGEMENT [Part 3, Section 2 Command Macro Descriptions]

TM — TASK MANAGEMENT [Part 3, Section 1 Command Macro Descriptions]

It is important to note that public volumes have been given attributes which tend to make them appear more as an extension of core storage rather than as conventional I/O devices. Their major attributes are summarized below:

1. The public volume is the normal (or default) type of volume.

2. A given file may be contained on any number of public volumes without the user's knowledge; in particular, it is the only type of volume on which a file can be automatically extended across volumes during execution of the problem program. Volumes associated with multivolume private files must be identified and provided for by the user.

3. Full file security and integrity are provided, yet the volume may be used concurrently by any number of tasks.

GENERAL

As pointed out in the preceding section, all user-initiated processing under VMOS takes place within the context of a task. Optimal use of the system's facilities by the programmer is, therefore, a function of his ability to control the task environment. The purpose of this section is to describe to the programmer those Command Language directives which provide for such control.

Although the control of processing takes place within the context of a task, not all Command Language directives relate to the control of the task itself. At the macro level in particular, the Command Language provides various program oriented instructions. As a result, it is necessary to discuss the control of processing from the standpoint of task and program management. In contrast to task and program, the term job has no special meaning in VMOS except in the context of remote batch processing. Use of the term in this document has, therefore, been restricted to discussions of this area of processing.

Unless the reader is directed to another manual or an appendix of this manual, all commands and macros referred in this section are fully described in the Task Management Section in Part 3 of this publication.

TASK AND PROGRAM INITIATION

As noted in the previous section, VMOS has established a hierarchy between tasks and programs by viewing the latter as a subset of the former. The commands supplied by the Command Language for initiating tasks and programs are, as a result, identified exclusively with one or the other. These commands are the means by which this system differentiates between a task and the programs processed within it.

Task Initiation

The programmer initiates a task when he successfully identifies himself to the system as a user. LOGON is the command provided by the Command Language for establishing this identification. A programmer cannot undertake any processing until the system has accepted his LOGON statement.

The information associated with the programmer's userid, as required in the LOGON statement, is checked against his JOIN Table entry when the programmer attempts to LOGON. If any discrepancy is noted as a result of this comparison, the interactive user is notified of the condition at his terminal; the background user is aborted.

The LOGON command, in addition to establishing the initial bounds of a task, also permits the programmer to exercise a degree of control over the task's environment. This control relates to task priority, alteration of message structure, buffer size stipulation, and CPU time requirements.

Task initiation of a limited nature is also provided for by the ENTER command. This command causes the execution of a file the contents of which define a task. However, like all other execution statements, the ENTER command must be issued from within an existing task. A complete description of ENTER files and the ENTER command is given under Procedure Files of this Section.

Program Initiation

Program initiation occurs when a user causes the loading and/or execution of a problem program such as: a user-written program, a software-supplied routine, or an application package. The commands that provide this facility are EXECUTE, LOAD, and DO.

The EXECUTE command enables the programmer to place a load module into memory and initiate immediate execution at the program's initial entry point. The program may be loaded from a file, an object module library, EAM space, or from SYSDTA. In addition, the EXECUTE command permits the programmer to specify the amount of additional memory required for his program and the maximum CPU time the program may use.

The LOAD command is functionally a subset of the EXECUTE command. It supplies the programmer with the same facility as does EXECUTE, with one exception. LOAD only places the specified load module into memory. To initiate execution of the program, the user must issue a RESUME command.

The DO command enables the programmer to call for the execution of a file which may itself contain one or more processing systems. A complete description of DO files and the DO command is given under Procedure Files of this section.

TASK REGULATION AND PROGRAM DIRECTION

VMOS, through Command Language facilities, provides a programmer with the ability to control task performance and direct the processing operations of user-written programs. The commands and macros which supply the means of implementing these facilities relate primarily to the functions of input-output processing, process monitoring, and the manipulation of the processing environment.

Task Regulation

Process regulation of the task level, within VMOS, is primarily a function of the way in which a programmer has structured his processing environment prior to task initiation. In essence, task regulation is equivalent to task management. In particular, however, the facilities provided for task regulation reside in the use of procedure files, the system's process interruption features, and LOGON-associated resource specification options, all of which are discussed separately elsewhere in this section. Nevertheless, the Command Language contains specific instructions for task monitoring and the specification of options applicable to the operation of language processors.

The programmer is able to monitor tasks using either the STATUS command or the TMODE macro. The STATUS command enables the programmer to have printed, at his terminal, information concerning the processing of tasks he has introduced. The programmer may obtain this information for all tasks bearing his userid or for a task identified by a specific task serial number. The STATUS command can only be issued in the interactive mode. The TMODE macro permits the programmer to have stored, in his user-written program, certain information about his task. The information which TMODE gathers relates to task type, terminal type (interactive tasks), task priority, task sequence number, the userid of the task initiator, the account number against which the task is charged, the CPU time used by the task, and the privilege code of the task. The programmer cannot obtain userid and accounting information for tasks not associated with his userid.

The PARAMETER command enables the programmer to specify source-input device options, output device options, work tape utilization, and debugging aid implementation for the various language processors available to him. A detailed explanation of this command is contained in the VMOS Programmer's Reference Manual.

Program Direction

In VMOS, the control of processing environment at user-written program level resides primarily in programmer's use of Command Language to write input-output directives, manipulate the processing environment, and bring about program-to-program communication. The Command Language instructions relating to input-output processing and process environment manipulation are presented in the following paragraphs; program-to-program communication is discussed separately later in this section.

Input Processing

The Command Language supplies the programmer with a pair of macro instructions for reading input from SYSDTA or SYSIPT, and a command to implement end-of-file processing. These instructions are the RDATA and RDCRD macros and the EOF command.

The RDATA macro provides the ability to retrieve data records from a task's SYSDTA file, i.e., the user's terminal, the system card reader, or a cataloged file on a direct access device. This instruction can also be configured to alert a user-written program that the assignment of SYSDTA has been changed. RDATA processes all data records as if they are variable format records. When RDATA is reading from a terminal and a break key interrupt occurs, the system returns control to the macro after the interrupt is processed.

The RDCRD macro has been supplied to provide TOS compatibility support and enables the programmer to retrieve data records from a task's SYSIPT file. SYSIPT may be the system card reader or a cataloged file on a direct access device. RDCRD expects the data record it reads to be in card image format and, it will truncate records greater than 80-bytes in length or blank-fill records less than 80-bytes long.

The EOF command is used in conjunction with the RDATA and RDCRD macros to provide end-of-file processing control. When SYSDTA or SYSIPT is not pointed to the same source as the task's SYSCMD file and RDATA or RDCRD reads a /EOF record, control is passed to the program's end-of-file routine. If SYSDTA or SYSIPT have the same source as SYSCMD, any command statement encountered by RDATA or RDCRD will cause control to pass to the program's end-of-file address. The only exception to this occurs when RDATA reads a /BREAK command: control is then returned to the next statement in the program after a /RESUME command is issued. The programmer is cautioned that once control has been passed to a program's end-of-file address, any subsequent reads to SYSDTA or SYSIPT from the program will also be directed to the end-of-file address. The EOF command may be used from a terminal in conjunction with the "BREAK" Key to transfer out of an interactive task loop. The VMOS EOF command supports the /* for TOS/TDOS compatibility.

Output Processing

The Command Language contains a set of five macro directives for the generation of system files and terminal destined output. Four of these directives (PROUT, WRLST, WRTOT, and WROUT) enable the programmer to write to the system output files; the remaining macro (WRTRD) provides the means for terminal to program communications.

WRLST and PROUT macros perform essentially the same function: enable a program to write records to the task's SYSLST file. In both cases, control is returned to the user-written program at normal termination of each write. When the execution of a PROUT results in a nonrecoverable error, the program is terminated, and control passes to the next STEP Command in the task or, if no STEP exists, to the task's LOGOFF command. (See Program-to-Program Communication for STEP command discussion.) Nonrecoverable errors associated with WRLST execution produce an error code for use by the program's error recovery routine. The PROUT and WRLST generated records are contained in SYSLST as V-type records. However, PROUT accepts records in fixed-length format which are then converted to variable-length format by the system; WRLST accepts variable-length records. The SYSLST file, containing the output, is spooled to the printer at task termination unless destined for a remote work station. After processing, the SYSLST file is erased. The PROUT macro has been provided primarily for TOS and TDOS compatibility.

User-written programs can write to a task's SYSOUT file through facilities of the WROUT macro. The SYSOUT file may be either a terminal or system work file. WROUT requires V-type records. If an interrupt occurs when WROUT is writing to a terminal, the system returns control to the macro so that the message can be rewritten. Normal termination returns control to the user-written program at the instruction following the macro. When a nonrecoverable error occurs, control is passed to the program at its error address. If SYSOUT is directed to a terminal and WROUT produces a record whose length exceeds the size of the terminal buffer, the message will be written to the terminal but the excess characters will be lost.

Records are written to the SYSOPT file using the WRTOT macro. Records acceptable to WRTOT cannot exceed 80 characters in length and must be in fixed length format. The task's SYSOPT file is spooled to the system card punch at task termination unless deferred for remote batch processing. Normal termination of WRTOT processing causes control to be returned to the program at the next instruction following the macro. An unrecoverable error causes program termination and control passes to the task's next STEP command or, if none exists, the task's LOGOFF command. The WRTOT macro is supplied primarily for TOS and TDOS compatibility.

Program-terminal intercommunication is supplied by the WRTRD macro. This macro enables a user-written program running as part of an interactive task to send messages to, and receive the required response from, the user's terminal. WRTRD processes only V-type records. Normal termination of WRTRD execution causes control to be passed to the next program instruction following the macro. A nonrecoverable error causes control to be passed to the program's error address. When WRTRD processing is interrupted, the system will cause the macro to be reexecuted.

Processing Environment Manipulation

Manipulation of the processing environment from within a user-written program is accomplished through use of the PASS, VPASS, and CSTAT macros. These macros enable the programmer to relinquish processor time and change the status of program pages.

The programmer relinquishes processing time by use of the PASS or VPASS macro. The principle difference between the two is the degree of control that a programmer is able to exercise. The VPASS macro enables the programmer to specify the period of time a task will relinquish its processor time. A time slice relinquished by the PASS macro is restored at system discretion.

The status of any one or all of the pages of a Class II may be changed using the CSTAT macro. This instruction enables a programmer to change the read-write access of the pages of a user-written program and also to designate program pages as nonresident (pagable) or resident. CSTAT cannot be used in a Class I program.

TASK AND PROGRAM TERMINATION

Similar to all other facets of task and program management, the termination of tasks and programs may be accomplished while operating in either the interactive or background environment. Furthermore, the facilities available to both modes of operation provide a programmer with the ability to halt the processing cycle prior to task or program completion, so that unexpected or undesirable conditions can be controlled.

Task Termination

In the normal course of events, the programmer terminates a task using the LOGOFF command. When the system receives LOGOFF, it causes processing for the associated task to cease, and returns the task's resources to the system. When operating in the interactive mode, a programmer can specify that a task be terminated but that the terminal remain attached to the system by using the BUT option of LOGOFF. This option allows the programmer to LOGON, and create a new task, without again dialing into the system.

The LOGOFF command, through use of the TAPE operand, provides the programmer with facility for having SYSLST, SYSOUT, and SYSOPT files written to tape. SYSLST and SYSOUT are written to the same tape; SYSOPT is written to its own tape. All these files are SAM files with variable length records. Subsequent to being spooled to tape, the files may be printed or punched as is appropriate to their origin.

Secondary tasks, such as those created by ENTER file processing, may be removed from the system prior to, or during, processing by using the CANCEL command. Primary tasks, however, cannot be terminated using CANCEL. This command also supplies a dump generating option. See Section 1 of this part for explantion of primary and secondary tasks.

The user can force a task to terminate abnormally by using the ABEND command. This command automatically forces a program dump of class 5 and 6 memory. ABEND terminates a task as if a LOGOFF had been issued. Like LOGOFF, terminal disconnect can be circumvented by using the BUT operand.

Program Termination

In addition to those commands which enable a programmer to halt processing at the task level, the Command Language contains a set of macro directives to facilitate program termination. These are TERM, TERMD, and TERMJ.

The TERM macro indicates the termination of the user's program. When a TERM macro is executed, the system deallocates all devices and memory associated with the program and returns control to the task's command stream. TERMD causes the occurrence of the same events as TERM, except that a program dump is also generated and written to the SYSLST file.

The TERMJ macro is provided to allow a programmer to indicate the termination of a step within a task. If the program containing TERMJ is contained within a background task, the next step in the task sequence (indicated by a STEP command) is initiated. However, if no more steps are to be run in the background task, TERMJ will cause the task to be terminated and control will be returned to the system. A TERMJ contained in a program running in a conversational task will cause the next command of task's command stream to be executed.

PROCESS INTERRUPTION

The VMOS Command Language contains command directives that enable the programmer to interrupt a task's processing environment at task level or at program level. The Command Language also contains a set of macro directives which provide for the control of task interruption processing. Since certain phases of task interruption are often related to interprogram communication, refer to the discussion of Program-to-Program Communication for more insight into VMOS' process interruption. The following discussion pertains only to direct program-to-system communication.

Task and Program Interruption

Task interruption is provided through use of the PAUSE command or the INTR (Interrupt) command. The PAUSE command enables a programmer to communicate with the operator either from an interactive or background task to have him perform some operation (put a card deck in a specified reader, mount a tape on a specific drive, etc.). PAUSE differs from the TYPE command (refer to Appendix B), which also can be used to communicate with the operator. This PAUSE command causes the user's task to be pended until the operator performs the requested operation. The operator reinitiates processing of the task.

The INTR command which can be entered either conversationally or nonconversationally, causes transfer to be passed to the operator's interrupt routine. This routine must be specified in the user's program through the use of a STXIT macro. Interactive tasks are interrupted by using an INTR at the terminal. Background tasks are interrupted by the operator issuing an INTR from his terminal along with the related task sequence number (TSN).

Program interruption can be accomplished using the BREAK command. This command can be entered as part of the command stream of a nonconversational task initiated from a terminal, or from the procedure of a DO file which has been initiated from an interactive task. When the system receives a BREAK command, control is transferred from the user's program to the system. The programmer may then enter command statements for the system to process. The programmer reactivates the execution of his program or procedure using the RESUME command. The RESUME command permits the user to resume processing at the point where the BREAK occurred or at a specified location within the processing stream.

Program Interruption Processing Control

Control, by the user, of the processing of program interruptions is supplied through a series of macros which relate primarily to the use of exit and contingency routines. These macros are EXIT, EXITP, SETIC, STXIT, and SPEXT. In the following discussion, interrupt routine and contingency routine are used interchangeably, since the programmer cannot control interrupts resulting from system operation.

EXIT and EXITP macros are concerned with the user's exit from, and return to, an interrupt or contingency routine. The EXIT macro enables a programmer to return an interrupt routine to the point in his main program at which the interrupt occurred. The EXITP macro enables the programmer to supply the information which will be used by the system to perform the functions that will permit a contingency routine to be reentered at the next applicable interrupt.

The SPEXT, SETIC, and STXIT macros relate to the construction of contingency or interrupt routines. SPEXT enables the programmer to have a Class II program obtain a status block containing information required by those interrupt routines available to the program. The SETIC macro allows the user to simulate an interval timer interrupt, and is used in conjunction with the STXIT macro. Finally, the STXIT macro enables the programmer to specify, within his program, the address of program interrupt routines for use by the system.

PROGRAM-TO-PROGRAM COMMUNICATION

Inter-program communication can be defined as the ability to pass information or control from one program to another during processing of programs or at completion of their processing sequences. The structure and use of procedure files, as discussed elsewhere in this section, describes one of the methods of program-to-program communication at the task level as a function of process control predefinition. The following discussion will center on those commands, relative to inter-program communication, that can be entered into the job stream independently or as sets, and on the macro directives that facilitate program-to-program communication from within the user's problem program. The perceptive reader will recognize that many of the commands discussed herein could be contained in a procedure file as well as a single-entry job stream.

Communication from Command Stream

The commands facilitating inter-program communication fall into two distinct sets: those that facilitate the passing of information from one program to another and those that pass control between programs. The commands that enhance a programmer's capability to pass information from program to program revolve around the use of CHANGE command.

The CHANGE command (CHNGE macro) enables the programmer to change the linkname of a file definition so that FILE command definitions can be passed from program to program within a task without changing the link symbols specified in the program's FCB. Thus, for example, a programmer can use the output file of one program as input to another without copying the file or changing its name. Further control is provided by DROP, HOLD, and RELEASE commands. The function of these latter commands is discussed in the File Management section.

The SETSW, SKIP, and STEP commands enable a programmer to pass control between programs. SETSW (and its equivalent macro) enables the programmer to set, reset, and invert the task switches provided by the system for conditional control of a task. By using SETSW in conjunction with the SKIP command, which provides the ability to test the settings of these task switches, the programmer can specify and control branching conditions within his program. The STEP command provides for logical subdivision of a task, and enables a programmer to control the effects of abnormal termination. When a program terminates abnormally, the system will pass control to the first command following the next STEP in the command input stream. If no STEP is encountered, control is passed to the task's LOGOFF command. If the programmer wishes to use a STEP command in an interactive task, it must be contained in a DO initiated procedure file.

Communication from within a Program

The macro directives relative to program-to-program communication enable the programmer to pass control between programs and to control their processing. The programmer can structure a user-written program so that it can read to and write from a common data area, call subroutines and load program overlays, manipulate task switches, and specify the loading and execution of other programs.

The EXCOM and TOCOM macros provide the means to access a common data area. EXCOM enables the programmer to have data from a common data area moved into a user-written program. TOCOM performs the reverse function. The common data area may be accessed by Class I and/or Class II programs. Data moved into the common data area will remain in the area for the duration of a session or until overwritten by another TOCOM. The common data area comprises 4096 bytes.

The CALL, LPOV, and RETRN macros provide the means of branching and performing overlaying operations from within a program's execution stream. The CALL macro enables a programmer to branch to a subroutine contained in the same load program. The RETRN macro provides the means of restoring the values of a program's registers prior to branching back from the subroutine to the main program. The values of the program registers should be retained prior to branching by use of the SAVE macro. LPOV enables the programmer to load a program overlay (in load module form) during program execution. If an LPOV is issued in a Class I program and an error results, the system terminates the task. In a Class II program, an LPOV associated error causes the system to place an error code in Register 15 and return control to the user's program at the instruction following the LPOV.

The Command Language provides for the manipulation of 32 task switches from within a user written program through use of the SETSW macro. This macro provides the same capabilities as the SETSW command, and is used in conjunction with a SKIP command entered as a part of a background task's command stream. In addition, the programmer has the ability to retrieve the settings of those switches using the SETSW macro.

The ability to initiate the loading or execution of a new load program (as opposed to a subroutine of the same load program) from within another is provided by the LOADM and EXECM macros, respectively. Both of these macros cause the program from which they are issued to terminate upon processing of the macro. After processing a LOADM, the system returns control to the task's command stream, from which a RESUME command must be issued in order for execution of the newly loaded program to begin. In all other respects, LOADM and EXECM macros provide the same facilities as do the LOAD and EXECUTE commands.

PROCEDURE FILES

VMOS provides the programmer with the ability to store Command Language commands in a file and have them executed as a set of instructions by calling for the processing of the file. Files so constituted are called procedure files. The principle benefit to be derived from the use of procedure files is that they enable a programmer to define repetitively-used operations in a way that eliminates the necessity of entering each command separately as it is called for by the requirements of the processing function.

In addition to commands, a procedure file may also contain data relevant to the processing activity defined within the file. For instance, a procedure file which contains the commands necessary for the execution of a program could also contain the input for the program. Conversely, the input required for a procedure may reside in a file other than the procedure file and be referenced from within the procedure file.

The programmer may create a procedure file as a DO file or as an ENTER file. The terms DO and ENTER are the names of the Command Language commands used to activate the file. The DO and ENTER commands are described in detail in Part 3, Section 1.

Procedure files may be created interactively using one of the VMOS File Editing Routines or by batch card input using the DATA command. Conventions relating to the use of File Editing routines are contained in the VMOS Programmer's Reference Manual as well as card input conventions (enumerated in Appendix D of the same manual). Other than the commands used to activate them, the principal differences between DO and ENTER files are the task and file delimiters peculiar to their structure, and the mode in which execution takes place. These and other salient features of DO and ENTER files are contained in table 2-6.

Enter Files

The ENTER file is a procedure file which, when activated, places a background task into the system's processing pool. ENTER files enable the programmer to initiate background tasks while operating in the interactive mode. After an ENTER file is activated, control is returned to the initiating task, thereby enabling the programmer to continue processing without waiting for that ENTER file to complete its procedure.

An ENTER file must contain all the commands necessary for the execution of a background task. The first record must be a LOGON statement and the last a LOGOFF statement. Because of independent processing, no communication is possible between the initiating task and the related ENTER file. When an ENTER file is activated, the system creates a separate, background task to be processed when resources become available. The sequence number for that task (TSN) is sent to the initiating task's SYSOUT file.

Similar to all VMOS files, ENTER files can be password protected at the file level. The read and write passwords (RDPASS, WRPASS) may be used to qualify file accessibility. Also, the processing of a file can be protected (ACCESS parameter of CATALOG command). If a file is read-password protected, the password (PASSWORD command) must be given before the file can be processed.

The programmer notifies the system when file processing is desired by giving an ENTER command and the filename of the related ENTER file; that is, /ENTER filename. When the ENTER command is issued, the user may also append to the filename all information generally associated with a LOGON command (userid, password, etc.). This information will then override all equivalent information associated with the LOGON command in the ENTER file. If no LOGON-associated information appears in the ENTER command, the system will use information from LOGON statement associated with primary task to override the equivalent information associated with LOGON command in the ENTER file. In effect, the user may specify that the processing of an ENTER be charged against a userid or account number other than the one associated with the primary task; by specifying the proper LOGON-associated information when he issues the ENTER command. However, if a userid is specified in the ENTER command, the programmer must also specify any other LOGON-associted information relating to that userid. If the programmer fails to supply this information, the system will not be able to find the proper JOIN Table entry, and the ENTER file will not be processed.

ENTER File Example

This example illustrates the operation and execution of an ENTER file from within an interactive task. The VMOS File Editor facility was used to create the file. Note that when the ENTER command was issued to activate the file, the programmer specified LOGON-associated information that was different from that associated with the ENTER file's LOGON statement. As a result, the charge for processing the ENTER file would be assigned to USERB, not USERA.

Initial LOGON and final LOGOFF statements define the extent of task (TSN) 6763. This includes the creation of file TJ.FILE and its activation by the ENTER command. The LOGON and LOGOFF statements of lines 100 and 1000, respectively, define the extent of processing of file TJ.FILE; that is, TSN 1093.

```
/LOGON USERA,,C'123'
%C E223 LOGON ACCEPTED AT 1601 ON 08/12/71, TSN 6763 ASSIGNED.
/EXEC (EDIT)
%P001 - DLL V-2A
 VERS. 0014 OF FILE EDITOR READY
*OPEN TJ.FILE
 %C T219 OPENED TJ.FILE AS NEW V-TYPEFILE.
*TEXT
      100.    <0>    /LOGON USERA,,C'123'
      200.    <0>    /PARAM LIST=YESS,DEBUG=YES,ERRFIL=YES
      300.    <0>    /SYSFILE SYSDTA=DATA.FILE.JIM
      400.    <0>    /EXEC BGFOR
      500.    <0>    /SYSFILE SYSDTA=(PRIMARY)
      600.    <0>    /EXEC TSOSLNK
      700.    <0>    PROGNAME MCG.TST
      800.    <0>    INCLUDE *
      900.    <0>    END
     1000.    <0>    /LOGOFF
     1100.    <0>    #END
*HALT
/ENTER TJ.FILE,USERB,,C'456'
      % TSN=1093
/LOGOFF
```

DO Files

DO file is a procedure file which, when activated, initiates the processing of a foreground (interactive) job. Unlike ENTER file processing, the system does not process the DO file as a separate task. The procedure is executed immediately upon being called, and additional processing within the calling task is suspended until the procedure has been completed. Therefore, when a DO file is initiated from an interactive task, the terminal remains attached to the task but control does not return to programmer until the procedure has finished processing. A DO file may only be a SAM file or ISAM file. The choice of SAM or ISAM depends upon the conventions of the medium used to create the file (File Editing Routines, DATA Command).

When a DO file is activated, it becomes the task's temporary SYSCMD file. The SYSDTA file remains directed to the initiating task unless directed to another source from within the procedure file. SYSDTA (or SYSIPT) may be directed to any source available to the SYSFILE command (see Part 3, Section 2). If SYSDTA is directed to SYSCMD, its source will be the procedure file; not the source associated with the initiating task. To cause SYSDTA to take its input from the source associated with the originating task, SYSDTA must be directed to PRIMARY. When the procedure terminates, SYSDTA (SYSIPT) and SYSCMD are redirected by the system, to the sources associated with the initiating task.

The first entry in a DO file must be a procedure statement (/PROCEDURE) and the last, an end statement (/ENDP). The remaining entries may be any commands acceptable to the system with the exception of LOGON, LOGOFF, and SECURE. LOGON, LOGOFF, and SECURE are considered task associated commands and a DO file cannot be constituted as a task. When one of these three commands is encountered in a DO file, the system sends a diagnostic message to the task's SYSOUT file, and continues processing with the next valid command in the procedure.

PROCEDURE Statement

The PROCEDURE statement supplies the user with a choice of printing options for the output resulting from the processing of the procedure, and enables him to relate symbolic parameters (entered with a DO command) to the processing of the procedure. Only one PROCEDURE statement is permitted in a DO file. A DO file may, however, contain calls to other DO files. Therefore, although a DO file may contain only one procedure, the procedure may reference other DO files. A more complete explanation of this phenomenon is contained in the description of Multiple Procedures. A PROCEDURE statement must be preceded by a slash. However, a PROCEDURE statement will not be recognized by the system when SYSCMD is receiving input directly from the task. In other words, the system only recognizes PROCEDURE statements in the context of DO files.

The format of a PROCEDURE statement and its operands are described and defined as follows:

Name	Operation	Operand
	{PROCEDURE / PROC}	[A / C / D / N] ,[(symbolic parameter-character,...)]

A

specifies that all records on the PROC file are to be written to the SYSOUT file as they are processed.

C

specifies that only command records on the PROC file are to be written to the SYSOUT file.

D

specifies that only data records on the PROC file are to be written to the SYSOUT file.

N

specifies that no records on the PROC file are to be written to the SYSOUT file. This is the default operand.

Symbolic parameter

an ampersand (&) sign followed by an alphabetic character, followed by zero to six alphanumeric characters. In order to represent a single ampersand sign within a character constant, two ampersand signs must be written. Any number of symbolic parameters may be combined in a given VMOS command. When a symbolic parameter is followed by a period, an alphabetic character or a numeric character, a period must separate the symbolic parameter from the character that follows. When a symbolic parameter is followed by a single period, the period is ignored. Symbolic parameters which appear in the comments field are ignored. Symbolic parameters are separated by commas and enclosed in parentheses.

Notes:

1. If the writing option (A, C, D, or N) is not specified and symbolic parameters are used, a comma must precede the first symbolic character.

2. For symbolic parameter processing, the PROCEDURE command is the analog of the DO command. Each positional parameter supplied in the DO command will be substituted for the corresponding symbolic parameter in the PROCEDURE command. The PROCEDURE command must contain at least as many positional parameters as are specified in the DO command. Keyword parameters supplied in the DO command are substituted for the corresponding keyword symbolic parameter. All keywords specified in the DO command must be present in the PROCEDURE command. Symbolic replacement occurs only in SYSCMD statements.

ENDP Statement

The ENDP statement returns control from a DO command initiated procedure file to the primary, command-input file. There may be, however, any number of ENDP statements in a DO file. This particular feature is covered in detail in the Multiple Procedures area of this Section. ENDP statements must be preceded by a slash and contain no operand field. Similar to the PROCEDURE statement, the ENDP will not be recognized by the system unless it is contained in a procedure file.

The same effect as the ENDP command can be created by the user by pressing the "Break" key at the terminal while an interactive task in the procedure mode is processing. However, in this case, the system prompts the user to verify that the "Break" key has not been pressed by mistake. After the User's action has been verified and the system has acknowledge the "Break" key, the procedure file cannot be resumed.

DO File Example

The following is an example of a DO file created using the VMOS File Editor. In this example, SYSDTA was directed sequentially to three different sources. At line 200, SYSDTA was redirected to a file containing a COBOL source program as an input source to the COBOL Background Compiler (BGCOB). Prior to this, SYSDTA was associated with the programmer's terminal. At line 400, SYSDTA was directed away from the COBOL source file to the procedure file (the task's temporary SYSCMD file) in order that the remaining commands in the file could be executed. Finally, at line 900, SYSDTA was directed to the terminal (its PRIMARY source) so that the COBOL load program (MCGRAW) could read its input from the terminal.

```
%C E222 PLEASE LOGON.
/LOGON USER-ID
%C E223 LOGON ACCEPTED AT 1452 ON 08/13/71, TSN 6871 ASSIGNED.
/EXEC (EDIT)
%P001 - DLL V-2A
 VERS. 0014 OF FILE EDITOR READY
*OPEN TJ.TEST
 %C T219 OPENED TJ.TEST AS NEW V-TYPEFILE.
*TEXT
      100.   <0>   /PROC C
      200.   <0>   /SYSFILE SYSDTA=TJCODE
      300.   <0>   /EXEC BGCOB
      400.   <0>   /SYSFILE SYSDTA=(SYSCMD)
      500.   <0>   /EXEC TSOSLNK
      600.   <0>   PROGRAM MCGRAW
      700.   <0>   INCLUDE *
      800.   <0>   END
      900.   <0>   /SYSFILE SYSDTA=(PRIMARY)
     1000.   <0>   /EXEC MCGRAW
     1100.   <0>   /ENDP
     1200.   <0>   #END
*HALT
```

Upon issuing the HALT command, the programmer was released from File Editor and control was returned to the operating system. When the system typed a slash at his terminal, the programmer activated the file TJ.TEST by issuing a DO command. The execution of this procedure file follows:

```
/DO TJ.TEST
%/PROC C
%00000200/SYSFILE STSDTA=TJCODE
%00000300/EXEC BGCOB
%L001 PROGRAM LOADING
 32A0 COMPILATION INITIATED (BGCOB VERSION=038A)
 32AA COMPILATION COMPLETED
%EB001 SPOOLOUT INITIATED FOR TSN=6880 ID=HPBLS857
%       PRINT FILE=00008
%00000400/SYSFILE SYSDTA=(SYSCMD)
%00000500/EXEC TSOSLNK
%P56D LOADING
 VERS. 008  OF TSOS LINKAGE EDITOR READY
 LIST? (Y, N)N
 PROGRAM BOUND
 PROGRAM FILE WRITTEN: MCGRAW

 NUMBER PAM PAGES USED:     6
%00000900/SYSFILE SYSDTA=(PRIMARY)
%00001000/EXEC MCGRAW
%P56D LOADING
 PLEASE REPLY WITH A 3 DIGIT NUMBER IN THE RANGE 0<N<330
*101
 THE NUMBER ENTERED WAS...101
 PLEASE REPLY WITH A 3 DIGIT NUMBER IN THE RANGE 0<N<330
*102
 THE NUMBER ENTERED WAS...102
 PLEASE REPLY WITH A 3 DIGIT NUMBER IN THE RANGE 0<N<330
*103
 THE NUMBER ENTERED WAS...103
 PLEASE REPLY WITH A 3 DIGIT NUMBER IN THE RANGE 0<N<330
*104
 THE NUMBER ENTERED WAS...104
 PLEASE REPLY WITH A 3 DIGIT NUMBER IN THE RANGE 0<N<330
*105
 THE NUMBER ENTERED WAS...105
%00001100/ENDP

/LOGOFF
```

Note: Because the file was created using File Editor, it is an ISAM file.

When constructing a DO file, the programmer need not be restricted to a single input scheme for execution of the commands contained within the procedure. Through use of symbolic parameters, the programmer can alter the processing of individual commands within the procedure by varying the input to them. In this way, a single procedure may be used to perform a variety of similar operations depending on the command input criteria specified when the file is activated. The ability to control the processing of a single procedure eliminates the need to construct separate procedures to perform functionally equivalent operations.

To construct a procedure that will accept symbolic parameters, the programmer must identify them in a PROCEDURE statement and in the command statements to which they apply. The character strings constituting symbolic parameters must be identical in both instances. The system correlates PROCEDURE statement entries with the parameter in command statements on the basis of their content; not according to their position in the parameter stream. Subsequently, when calling for execution of the file, the programmer enters, with the DO command, the input parameters desired to be associated with the symbolic entries in the file's PROCEDURE statement. The way in which symbolic parameters are constructed is covered in the discussion of the PROCEDURE statement.

The following illustrates one of the ways symbolic parameters may be used to generalize a procedure. This illustration is based on the previous DO file example in which a COBOL source program was compiled, linked, and then executed.

```
/LOGON
%C E223 LOGON ACCEPTED AT 1130 ON 08/16/71, TSN 6956 ASSIGNED.
/EXEC (EDIT)
%P001 - DLL V-2A
 VERS. 0014 OF FILE EDITOR READY
*OPEN TJ.TEST
 %C T219 OPENED TJ.TEST AS NEW V-TYPEFILE.

*TEXT
      100.   <0>   /PROC C,(&A01,&B02)
      200.   <0>   /SYSFILE SYSDTA=&A01
      300.   <0>   /EXEC &B02
      400.   <0>   /SYSFILE SYSDTA=(SYSCMD)
      500.   <0>   /EXEC TSOSLNK
      600.   <0>   PROGRAM MCGRAW
      700.   <0>   INCLUDE *
      800.   <0>   END
      900.   <0>   /SYSFILE SYSDTA=(PRIMARY)
     1000.   <0>   /EXEC MCGRAW
     1100.   <0>   /ENDP
     1200.   <0>   #END
*HALT
```

As in the previous case, the procedure file was created using File Editor. However, this time, symbolic parameters were entered as operands of the SYSFILE command (line 2000), the EXEC command (line 300), and the PROCEDURE statement. In the SYSFILE and EXEC commands, these parameters replaced the name of the COBOL source file and the name of the background compiler, respectively. The symbolic parameters must appear in the PROCEDURE statement to supply the link to the DO command. The procedure was executed as follows:

```
/DO TJ.TEST,(TJCODE,BGCOB)
%/PROC C,(&A01,&B02)
%00000200/SYSFILE SYSDTA=&A01
%00000300/EXEC &B02
%L001 PROGRAM LOADING
 32A0 COMPILATION INITIATED (BGCOB VERSION-038A)
 32AA COMPILATION COMPLETED
%EB001 SPOOLOUT INITIATED FOR TSN=6983 ID=HPBLS857
%       PRINT FILE=00005
%00000400/SYSFILE SYSDTA=(SYSCMD)
%00000500/EXEC TSOSLNK
%P56D LOADING
 VERS. 008 OF TSOS LINKAGE EDITOR READY
 LIST? (Y, N)N
 PROGRAM BOUND
 PROGRAM FILE WRITTEN: MCGRAW

 NUMBER PAM PAGES USED:     4
%00000900/SYSFILE SYSDTA=(PRIMARY)
%00001000/EXEC &C03
%P56D LOADING
 PLEASE REPLY WITH A 3 DIGIT NUMBER IN THE RANGE 0<N<330
*101
 THE NUMBER ENTERED WAS...101
 PLEASE REPLY WITH A 3 DIGIT NUMBER IN THE RANGE 0<N<330
*102
 THE NUMBER ENTERED WAS...102
 PLEASE REPLY WITH A 3 DIGIT NUMBER IN THE RANGE 0<N<330
*103
 THE NUMBER ENTERED WAS...103
 PLEASE REPLY WITH A 3 DIGIT NUMBER IN THE RANGE 0<N<330
*104
 THE NUMBER ENTERED WAS...104
 PLEASE REPLY WITH A 3 DIGIT NUMBER IN THE RANGE 0<N<330
*105
 THE NUMBER ENTERED WAS...105
%00001100/ENDP
```

Note that had the programmer entered the name of a properly constructed ANSI COBOL program and the name of the ANSI COBOL compiler in lieu of TJCODE and BGCOB, the execution of the procedure would not have been altered. In this case, the symbolic parameters related to what was executed, not the sequence of execution.

This example was not intended to be definitive, but to introduce the reader to the use of symbolic parameters in a DO file.

Multiple Procedures

VMOS does not restrict the programmer to DO files using symbolic parameters as the only means of altering the execution of a procedure. The system permits the construction of procedure files which, in turn, call for the activation of other procedure files. This facility is not restricted by file type: for example, an ENTER file may contain calls to DO files as well as to other ENTER files. The same holds true for DO files. In addition, the number of additional procedures which can be entered from the initial procedure file is functionally limitless. Both DO and ENTER files can contain DO and ENTER files which, in turn, contain DO and ENTER files, and so on.

In a DO File

Any DO file may contain, within its procedure, calls to other DO files or ENTER files. This is true whether the file containing the call is the initial procedure file or a subsequent procedure file activated from within the initial file. As an illustration, consider the following generalized example of a DO file which calls one of two other DO files, the last of which calls, in turn, an ENTER file. Each of the three files are constructed separately as follows:

DO File OUTER	DO File INNER1	DO File INNER2	ENTER File BACKGRD
/PROC C,(&DO2)	/PROC C	/PROC C	/LOGON Userid
.	.	.	.
.	.	.	.
.	.	.	.
/DO &DO2	.	/ENTER BACKGRD	.
/ENDP	/ENDP	/ENDP	/LOGOFF

Note that each of the preceding files meets the criteria of a self-contained procedure: the DO files start with PROC statements and terminate with ENDP statements; the ENTER file starts with a LOGON command and terminates with a LOGOFF.

The order and choice of execution for each procedure is displayed on figure 2-1. As shown, the preceding flow of execution illustrates three important features relative to the construction and processing of multiple procedures: the use of symbolic parameters, the termination of the procedures, and the subsequent transfer of control.

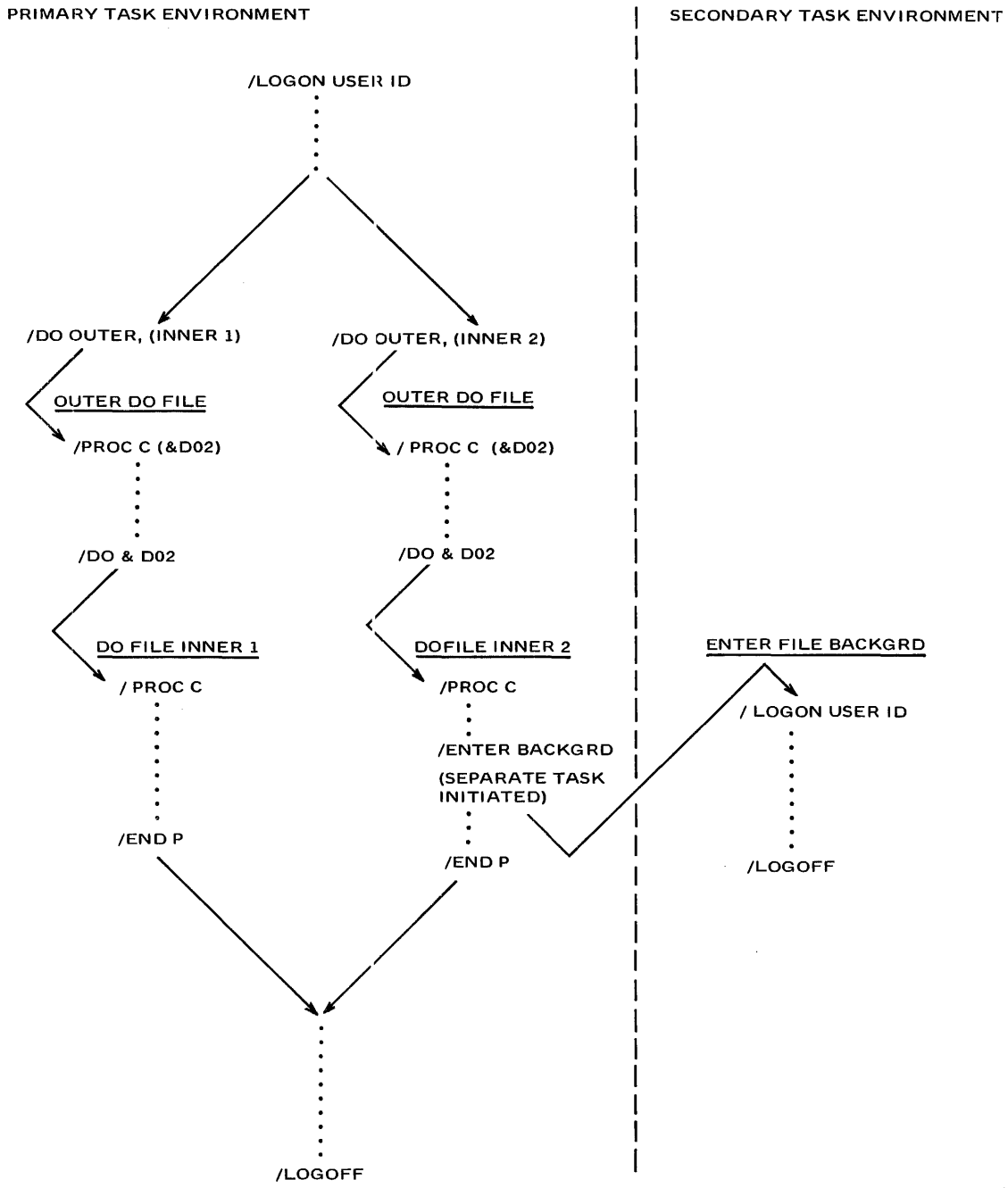


FIGURE 2-1. DO FILE PROCEDURES

The use of symbolic parameters permitted the construction of a procedure (OUTER DO File), from within which one of two other procedures (INNER1 and INNER2) could be called. The choice, relating to which of these inner procedures is to be executed, is made by the user when the outer procedure is called (DO OUTER, symbolic parameter).

Each of the three DO file procedures terminate in one of two ways: by an ENDP command or by a call to another procedure. This is the only way in which DO file procedures may terminate in a normal fashion. The procedure in a DO file will terminate at an ENDP statement unless a call to another procedure (DO or ENTER command is encountered prior to the ENDP). None the less, an ENDP statement must always be the last entry in a DO file procedure when the procedure is constructed, regardless of how the procedure is expected to terminate.

The termination of a DO file procedure by a call to another DO file causes control to pass from the calling procedure to the called procedure. The termination of a procedure by an ENDP statement causes control to pass to the initiating task regardless of whether or not the procedure was called from within another DO file procedure. Control can never be returned to a preceding DO file procedure by a subsequent DO file procedure. Note, however, that the initiation of an ENTER file from within a DO file procedure does not cause any transfer of control. The system sets up a separate background task to process the ENTER file procedure but control remains with the DO file procedure which called the ENTER file.

In an ENTER File

As in the case of DO file processing, both DO and ENTER files can be called from within an ENTER file. However, the processing sequence is less complex. Because the initiation of an ENTER file always creates a separate background task, the advantages of initiating a DO file reside primarily in the user's ability to specify a tangential procedure, rather than in his ability to effect optional procedure selection, since he cannot control execution using symbolic parameters. Nevertheless, an ENTER file is never terminated by a call to another procedure. If the called procedure is a DO file, or a series of successively called DO files, control will return to the ENTER file which initiated the first call to a DO file when the last ENDP statement is processed. An ENTER file procedure may also call another ENTER file. In this instance, however, control remains with the calling procedure and processing continues as soon as the system has set up the background task to process the second ENTER file.

Table 2-6 lists the salient features of the DO and Enter files.

TABLE 2-6. PROCEDURE FILES

	ENTER File	DO File
File Activation	The ENTER (filename) command	The DO (filename) command
Procedure file treated as separate task?	Yes	No
File delimiters	LOGON command LOGOFF command	PROC command ENDP command
Task delimiters	LOGON, LOGOFF commands in the procedure file.	LOGON, LOGOFF commands in the initiating task.
Can be initiated from an ENTER file?	Yes	Yes
Can be initiated from a terminal?	Yes	Yes
Can be initiated from a DO file?	Yes, but control remains with the calling DO file and processing continues after system creates the background task to process the ENTER file.	Yes, but when the ENDP command is encountered, control returns to original initiating task.
If the procedure is initiated from a terminal, does the terminal remain connected to the procedure?	No	Yes
Action that occurs due to terminal initiation.	The system types TSN of background task and returns a slash to the terminal. User can then LOGOFF if desired, thus saving telephone charges, while the task processes in background.	Initiated procedure is part of the initiating task. The system returns a slash when procedure is completed.

REMOTE BATCH PROCESSING

The VMOS Remote Batch Processing (RBP) System provides the programmer, at a remote terminal, with the ability to use the full batch-processing potential of the operating system. The programmer using the RBP system can enter card-input jobs into the system, monitor their progress, cancel them, and redirect the output of a finished job at task termination. With the exception of the RJOB, RSTATUS, and RMSG commands, all commands controlling RBP may be entered only from a remote work station (terminal); if entered from the control card reader, they will be rejected by the system.

Remote Job Initiation

Once a remote work station has been physically attached to VMOS, the RBP system will place the device in the inactive state and begin monitoring it for input. The user logically attaches the work station to an RBP system by issuing the RSTART command. RSTART must be the first command issued at the beginning of each remote batch session, as it identifies the remote work station to the RBP system. After an RSTART is received and validated by the RBP system, the work station is considered to be in the active state.

Work stations in the active mode are ready to receive RBP input. The programmer begins the processing session by issuing an RLOGON command. RLOGON functions in the RBP environment the same way LOGON functions in the local batch and interactive environment. The RLOGON command, like the LOGON command, defines the beginning of a task, and must precede all processing input. The information supplied with RLOGON command is checked against the user's Join Table entry by the RBP system, and must agree before the RBP system will accept the user's program input. The RLOGON command remains in effect until an RLOGOFF, RSTOP, or subsequent RLOGON is received by the RPB system from the work station. If a valid RLOGON is received from a work station with a session in progress, the current user is logged off and a new user is logged on.

Remote Job Regulation

The RJOB command enables an RBP system user to identify jobs within his task. By using the RJOB command, the programmer has the ability to defer job output until it is explicitly requested (see ROUT command) and to designate an alternate output recipient.

When a job identified by the RJOB command has finished executing, the submitter and alternate output recipient (if one exists) are automatically notified of the completion. Deferred job output may then be retrieved at any time. If the user does not identify his input using an RJOB command, the RBP system will supply a jobname and direct output to the submitter after job completion.

The RJOB command may also be used to define RBP for jobs submitted from a central installation card reader. This provides the programmer with ability to direct output from centrally submitted jobs to user's at remote work stations.

The ROUT command allows the RBP user to retrieve deferred job output (see RJOB, DEFER option) for which he is a valid recipient. If the ROUT command is issued and the job specified is not completed, the command is ignored and a message is sent by the RBP system to the requestor noting that the job is not completed. The ROUT command must then be reentered at completion of the task. If a ROUT command is issued for a specific job where the user is not a valid recipient or for a job which is no longer in the system, the RBP will return a message to that effect.

The ROUT command also allows the user to retrieve output that was discontinued due to user intervention or by equipment failure. When interrupted output is pending for a work station, no output is returned to the work station until a ROUT command determines disposition of the interrupted job. However, the system will continue to accept input from that work station. If the work station has output pending and is logically detached from the system (via RSTOP command), the next time a user attaches that station (via RSTART command), the job output will be returned for starting at the point it was interrupted. If the output was interruped by a transmission failure, the user must reattach the work station via the RSTART command.

In addition, the ROUT command enables a user to delete jobs from the RBP system without receiving a copy of the job output. The user can delete only those jobs that have been submitted and are still currently in the system. The system will return an invalid request response if the user attempts to delete jobs that do not belong to him.

Finally, it should be noted that if a ROUT is entered without modifying parameters, the system will resume output for any job, submitted from that specific work station, that is in a discontinued state. If job output is not in a discontinued state, this form of the ROUT command merely resets the work station to a state capable of receiving output.

The RMSG command enables a user to send messages to another user, the console operator, or an RBP work station. Such messages will be rejected if they are directed to inactive users (see RLOGON command) and/or work stations not logically attached to the system (see RSTART command).

The RSTATUS command provides the user with the ability to determine the status of RBP jobs for which he is a valid output recipient. RSTATUS enables the user to obtain the status of a specific job, the status of jobs submitted by a specified user, or for all current jobs submitted from a work station. The RBP system returns the status of only those jobs that are in the system when the command is processed. The information a programmer receives in response to an RSTATUS comprises the jobname, submitting userid, submitting terminal id, alternate user, and whether the job is executing, complete, or not in the system.

Remote Job Termination

The RLOGOFF command is the last command contained in a remote batch processing task. The programmer enters RLOGOFF to indicate that he has completed the processing session. After the receipt of an RLOGOFF, the RBP system will not accept input from the user's work station until a new session is begun (RLOGON command). However, the work station, because it is still logically attached to the RBP system, will continue to monitor the system for output.

The RSTOP command enables the programmer to detach a work station from the RBP system. When this command is entered, the RBP system transmits all queued messages to the terminal that is being detached prior to the actual detachment. However, no job output will be returned to the work station after the RSTOP command is processed. The last message transmitted by the RBP system will indicate that the work station is detached from the system. No further communication occurs until the work station resumes RBP activity with an RSTART command. If the work station is connected to VMOS via a dialed connection, the connection is broken.

CHECKPOINT/RESTART FACILITIES

The VMOS Checkpoint/Restart Facilities enable the batch programmer to specify points within a program at which the system environment and program status are to be captured, and then to have execution resumed at these points. The user designates and maintains control of the file to which these checkpoints are written, thus allowing the best combination of file space economy and restart capability that will meet particular needs. Insofar as is practical, restarting is flexible: both as to mode of invocation and to the ease of program modification prior to reexecution.

Checkpoint/Restart operates under control of the VMOS Control Program and is initiated using directives supplied by the Command Language. The Checkpoint facility is invoked by an SVC generated when the programmer enters the CHKPT macro (see Part 3, Section 1). This feature monitors the system environment in relation to the program as the environment exists when the program processes the CHKPT.

The programmer invokes the restart facility by issuing the RESTART command (see Part 3, Section 1) from a terminal or as part of a spooled-in job stream. This facility causes the program to be reloaded and execution to begin from the location at which the specified checkpoint was taken. Execution of the program can, however, be suppressed by the programmer.

The information required to restart a program may either be specified by the user or obtained from a log created by the system when the checkpoint is taken. The system creates the checkpoint file to contain this information as a Primary Access Method (PAM) file. This file provides the medium wherein the program and its environment, as recorded by the checkpoint operation, is retained for use by the restart facility.

Public or private volumes may be used to contain the checkpoint file. If private volumes are used, the programmer must assume that device on which they are contained is supported by PAM. In other words, paper tape cannot be used as a medium for the checkpoint file; magnetic tape and direct access devices can, as they are supported by PAM.

Message Generation

Checkpoint writes one message to SYSOUT, to log a successful checkpoint:

CE301 CKPT: mm/dd/yy hh tt xxxxxxΔHFPG=p

mm/dd/yy	is the current date
hh tt	is the time (hours and minutes)
xxxxxx	is the checkpoint ID
p	is the page number on which the checkpoint header is written

The same message is also written to the operator's console for nonconversational tasks.

RESTART may write one, two, or three of thirteen possible messages to SYSOUT. These messages are contained in the VMOS Messages Reference Manual.

Considerations for Use

The following considerations comprise usage conventions and restrictions imposed by either the Operating System or the Checkpoint/Restart facilities themselves. The user is advised, therefore, to pay particular attention to these points.

Stack Management

The Checkpoint/Restart philosophy assumes that the program issuing the checkpoint SVC is running with just one P1 stack (its own), one P2 stack (Command Language Processor), and was given control as the result of a /EXECUTE, /RESUME, or /RESTART command. Consequently, a checkpoint taken in any other environment will probably be unusable as input to the restart process. Note that this also applies to subsequent checkpoints taken by a restarted program. For this and other reasons, the /RESTART command must not have been issued from a more complex environment. In particular, it may not be issued from a Procedure File.

This restriction does not concern temporary system stacks (Break or BROADCAST stacks, for example) but rather is a constraint on the user's design.

File Management

The user must be extremely careful that the file to which checkpoints are to be taken is opened in such mode as will achieve the desired results. In effect, by the frequency of the opens and closes and by the mode in which a file is opened, the user specifies how often (if at all) checkpoints are to be overwritten and when retained. It should be noted, also, that failure to allow secondary allocation would cause one or more checkpoints to be aborted, if there is not sufficient file space remaining to contain it.

File Integrity

Although every effort will be made to assure the integrity of the checkpoint file, the user is advised not to use it for any other purpose; that is, it should be a dedicated file. Interspersing data-records with checkpoints on the same file is not recommended. Data files opened OUTPUT or OUTIN cannot be saved by Restart.

EAM Files

Evanescent Access Method (EAM) files, including SYSLST, SYSOPT, and SYSOUT, are not checkpointed and cannot be reconstructed at restart time.

BTAM Files

BTAM files are not repositioned in Restart.

Restart Environment

If a /RESTART command is issued while a program is already loaded in the user's virtual memory space (for example, from a previous /LOAD command or a /EXEC followed by a BREAK), the user program will be terminated automatically before restarting commences.

The /RESTART command may not be issued from a PROCEDURE file.

A /RESTART command may not be issued while SYSDTA is redirected to a secondary file.

A /RESTART command may not be issued while SYSIPT is redirected to a secondary file.

Secondary SYSFILE Assignment

A checkpoint, taken while SYSCMD, is under secondary assignment to the Card Reader cannot be restarted.

Checkpoint/Restart Optimization

The primary area of control which the user has over the time consumed in taking a checkpoint is in the allocation of file space for the checkpoint file. The number of PAM pages used for a single checkpoint may be approximately calculated by the formula: P = 2n + 6 where n is the number of pages of virtual memory currently (at checkpoint time) allocated to the program.

If possible, the checkpoint should be taken at a point in the program where the amount of class 5 and class 6 memory allocated to the program is minimized. For example, if memory is requested and released in a program on a frequent and dynamic basis, the user is best advised to checkpoint when the requested memory is at a minimum.

The second consideration involves making the initial allocation of file space large enough to accommodate all expected checkpoints, thus avoiding secondary allocations. If this is not feasible, as is often the case, the next best alternative is to be sure that the secondary allocation specified equals or exceeds the requirements of any one checkpoint as calculated by the above formula. In order to inhibit allocation of class 5 and class 6 memory while a checkpoint is being taken, the checkpoint module will force secondary allocations (sufficient to contain the checkpoint) by dummy writes in advance of the checkpointing writes. Since one write accompanies each allocation, a minimum secondary allocation (3 PAM pages) can be expensive.

By choosing the manner and timing of the way in which he opens and closes his checkpoint files, the user may accomplish his particular objectives within minimum file space. The examples on figure 2-2 through 2-5 are intended to be illustrative, but not exhaustive. The reader will doubtless be able to devise many other schemes.

On figure 2-2, all checkpoints are written consecutively to a single file. Restart may be accomplished from any one of them, but a great deal of file space may be used.

In figure 2-3, the file space is minimized, but only the last checkpoint is ever available and even that would be destroyed in case of a malfunction during the last checkpoint.

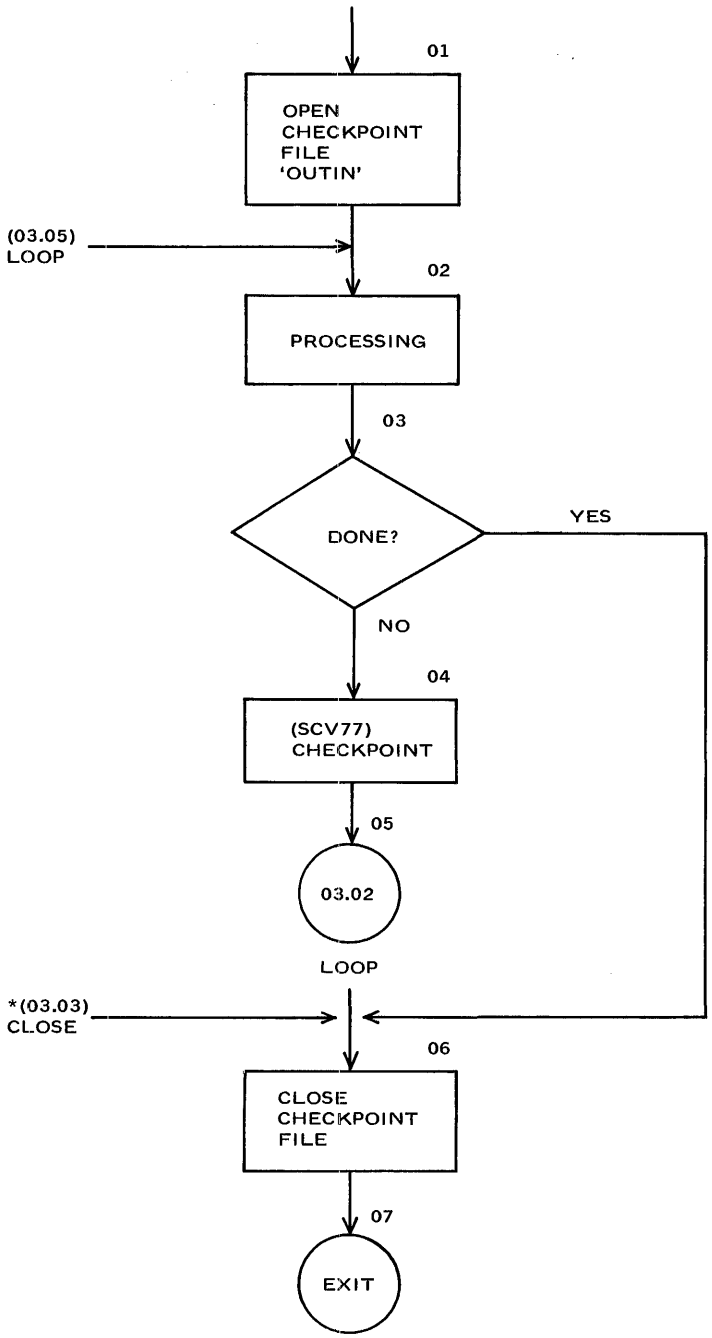


FIGURE 2-2. SINGLE SERIAL FILE

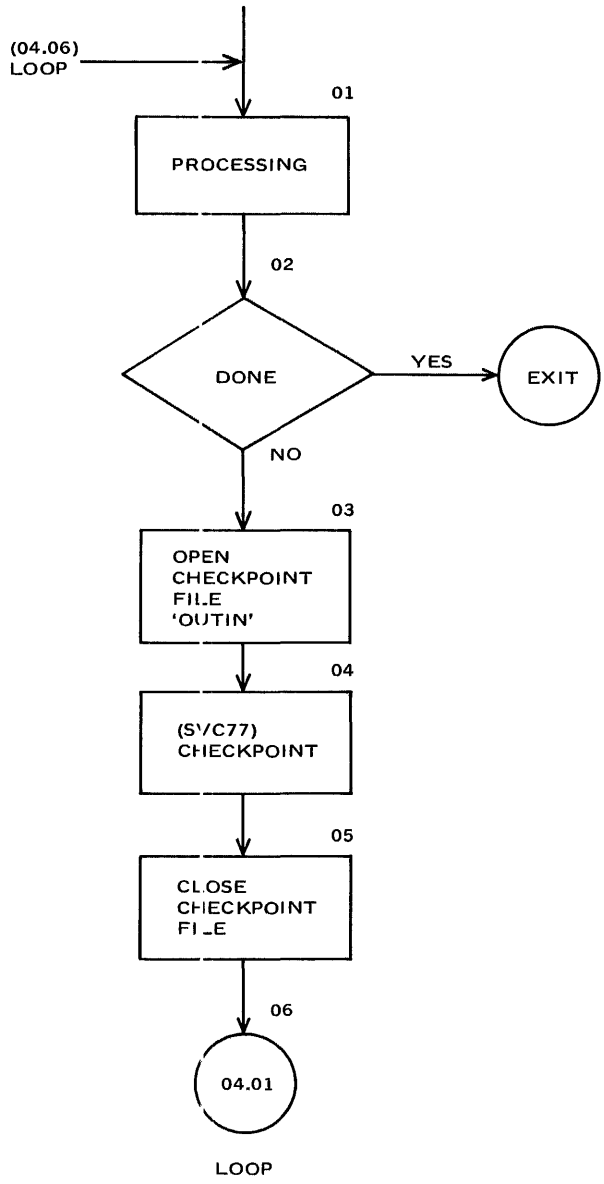


FIGURE 2-3. SINGLE CHECKPOINT

The illustration given in figure 2-4 will ensure that at least one checkpoint is always on file.

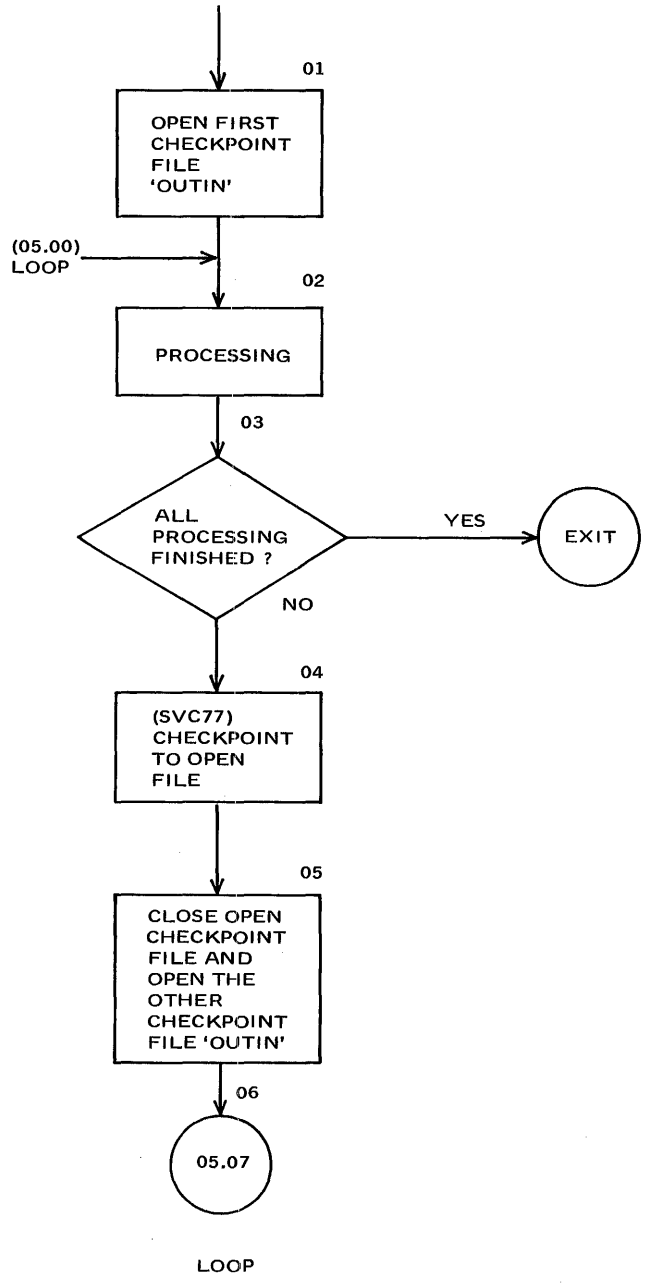


FIGURE 2-4. ALTERNATING CHECKPOINT

The contents of figure 2-5 would allow restarting at either one of the last 'n' checkpoints (end of volume, etc.) or at any of a series of major checkpoints. This allows restart in case of malfunction or for file reconstruction purposes at less than maximum file allocation.

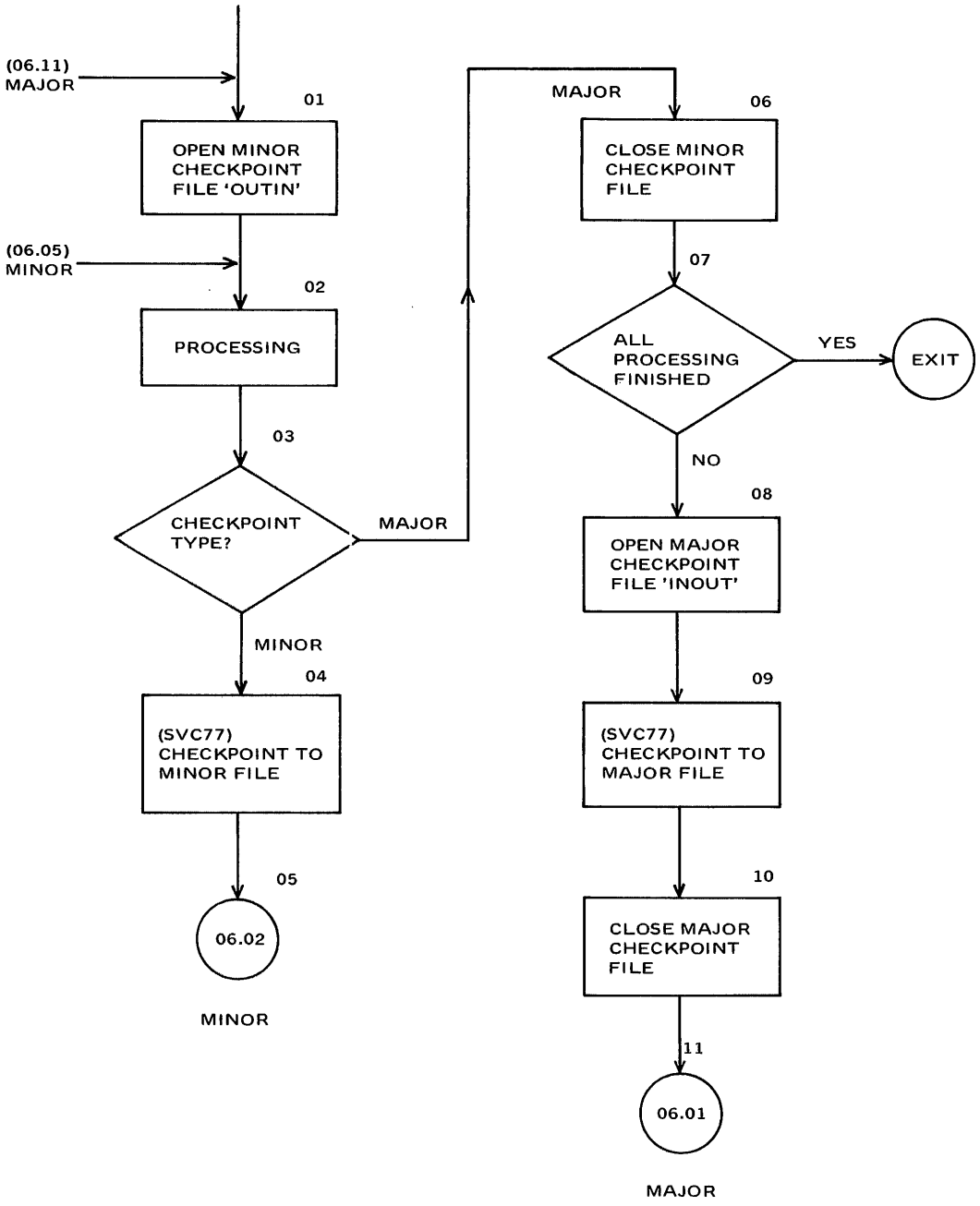


FIGURE 2-5. MAJOR AND MINOR CHECKPOINTS

The VMOS Checkpoint/Restart logic subsumes both Class I and Class II programs. However, some Class I programs may be TOS programs and require that checkpoints be taken by issuing a TOS CKPT macro (SVC 14). The purpose of this section is to describe the steps necessary to interface such a supervisor call with the logic for VMOS checkpoints. It should be noted that no additional logic is necessary for restart since TOS/VMOS program differences are not significant in restart. Note that the user must supply restart file pointers in Class I checkpoint.

Prior to executing a TOS Program which issues the CKPT SVC, a VMOS user must define the checkpoint file by using a FILE command with a LINK name of CKPTFILE. The device, if specified, may be any for which PAM is supported. The FCBTYPE and OPEN parameters should not be specified.

The space allocated is subject to the same restraints as for VMOS checkpoints, with one exception. If secondary allocation is not allowed, that is, SPACE=(n,0) has been specified, and if the primary allocation is great enough to accommodate at least the largest checkpoint, no error will be returned. If sufficient space is not available to accommodate all checkpoints, checkpoints will be rewritten starting from the beginning of the file.

When a CKPT SVC is issued by a Class I program, the TFT entry is located, an FCB is created and opened, and a VMOS CHKPT parameter list is built. The VMOS checkpoint logic is then executed. TFT and FCB are explained in the File Management Section which follows.

On subsequent CKPT SVC's, the FCB is located, parameter updating done as required, and a rough check for sufficient remaining space is made. Any necessary provision for space is made before joining the CHKPT logic.

At program termination, the checkpoint file will automatically be closed. This allows the user to issue commands between programs affecting the file or TFT entry. This allows reuse of the file for succeeding checkpoint files or reuse of the link name for writing checkpoints from a successor program to a different file.

GENERAL

The scope of those facilities supplied by the VMOS Control Program enables the programmer to create, modify, and dispose of program files, and to manipulate certain of the System Logical files. Before exploring these areas of file management, the basic principles – relating to file naming, file security, file retrieval, and the structure of catalog blocks – require explanation.

Unless the reader is directed to another manual, all commands and macros referred to in this section are fully described under File Management Commands in Section 2 of Part 3 of this publication.

Filenames

All names of user files created under VMOS must conform to the following format:

[$userid.]name[.name...][(group)]

The programmer, when he names a file, is responsible for the

name[.name...][(group)]

portion of the full filename. The $userid is supplied by VMOS automatically at file creation, and corresponds to the user identification of the file's creator. By prefixing filenames with the userid of their creators, VMOS assures that all references to a file will be directed to that portion of the system catalog associated with the correct user.

The entire filename (including $userid) may contain no more than 54 characters. The programmer-supplied portion, name[.name...][(group)], may not exceed 44 characters. Description of each of the filename components is as follows:

userid

Specifies the programmer's identification code. The maximum size is 10 characters including the period and dollar sign.

name

Specifies a simple configuration of one or more characters. The characters may be alphanumeric or hyphens, but neither the first nor last character can be a hyphen. However, a simple name may contain more than one hyphen. Simple filenames, not part of a compound name, may contain more than 8 characters.

Compound names may be formed by connecting one or more simple names using periods (for example, PAYROLL.MASTER, PAYROLL.MASTER.JIM, etc.). The rules for forming the component of a compound name are the same as those applying to simple names. The first simple name in a compound name may not exceed a length of 8 characters.

(group)

Specifies the identification of a member of a set of historically related files. This identification may consist of a simple name or a compound name, and the aforementioned rules for forming names apply. Note, however, that group identification must be enclosed in parentheses.

Filename Qualification

The ability to append qualifiers to the "name" portion of a filename supplies the VMOS user with a convenient method for grouping files into meaningful categories. Thus, all the payroll files for a particular application could be grouped together under the basic name PAYROLL as follows: PAYROLL.MASTER, PAYROLL.TRANSACTIONS, PAYROLL.UPDATE(1), etc. This facility results from the implementation in VMOS of the concept of partially and fully qualified filenames.

The filename as specified when the catalog entry is created is the fully qualified filename, and can be a simple or compound name. If a filename is compound, it may be identified by omitting the rightmost simple name or names. The remaining name (simple or compound) plus a trailing period is the partially qualified filename. For example, MATH.TRIG. is the partially qualified filename identifying the filename MATH.TRIG.COSINE. As mentioned previously, a partially qualified filename, such as PAYROLL., can be used to identify a group of related files, such as PAYROLL.MASTER and PAYROLL.TRANSACTIONS. The partially qualified filenames are accepted by the system for processing as parameters of the FSTATUS (file status) and ERASE instructions.

Because VMOS associates all filenames with the userid of a programmer who created the catalog entry, only that programmer can reference the catalog entry using the fully qualified or partially qualified filename. Any programmer, not identified to the system by the userid of one who created the filename, must enter the fully qualified filename prefixed by userid of the creator. In other words, userid 1 cannot access filename XYZ created by userid 2 unless he identifies the filename as userid2.XYZ. Userid 2 must also have created file XYZ as sharable (see File Security description).

Examples

Userid 1 created the following files as sharable: A.B.C,A.B.D,A.E,X.Y,X.Y.Z, and ENCYCLOPEDIA. User 2 may specify any one of them by prefixing the name with userid1: for instance, Userid1.A.B.C.

The following partial to full filename correspondence is also true.

A. specifies the files A.B.C,A.B.D, and A.E

A.B. specifies the files A.B.C and A.B.D

A.B.C specifies only filename A.B.C

A.B.C. specifies none of the named files, as no 4-character filenames beginning with A.B.C. exist.

A specifies none of the files (A≠A.)

X.Y specifies file X.Y

X.Y. specifies file X.Y.Z (not X.Y)

File Groups and Renaming Tapes

The VMOS facilities normally invoked for renaming direct access files associated through the use of the (group) entry in the filename (see CATALOG command) do not suffice for tape files. The renaming of tape files requires the modification of the file label, and the nature of tapes is such that the rewriting of a block in effect destroys all blocks beyond it. Therefore, if a tape label was rewritten, all the remaining data on the tape would be functionally destroyed.

In order to circumvent this problem, the following convention has been established. If the name of a tape file contains a left parenthesis, the left parenthesis and all subsequent characters in the name are not output in the tape label. Typically, the user would suffix the character string (integer) to the filename.

Examples

Example 1

Filename	Name in catalog. Name used to reference the file in commands, macros, FCB.	Name in the label of the tape volume.
PAYROLL	PAYROLL	PAYROLL
PAYROLL(1)	PAYROLL(1)	PAYROLL
PAYROLL.A(1)	PAYROLL.A(1)	PAYROLL.A
PAYROLL.(1)	Illegal	

When a tape file is processed and the name of the file in the catalog contains a left parenthesis, the left parenthesis and all subsequent characters are ignored in the tape label comparison for that file.

Example 2

Assume two copies of a payroll file entitled:

PAYROLL (current version),volume serial number (vsn)35
PAYROLL(1) (backup) , vsn 50.

The user wishes to create a new version of the PAYROLL file and to cause the current version to become the backup version. The following commands could be used:

```
CATALOG PAYROLL(1),DUMMY,STATE=UPDATE
CATALOG PAYROLL,PAYROLL(1),STATE=UPDATE
CATALOG DUMMY,PAYROLL,STATE=UPDATE
```

These commands could, of course, be retained in a procedure file to facilitate use of this renaming procedure. In the first command example, PAYROLL(1) is temporarily changed to DUMMY, since in the next command a new PAYROLL(1) is defined. After execution of these commands, PAYROLL references the tape with a vsn of 50, and PAYROLL(1) references the tape with a vsn of 35.

Linkname

Linkname is the term applied to the file reference variable that provides connection between the File Control Block (FCB) macro and the Data Management System's Job Control Language. The linkname ties together the programmer's FILE command, the file itself, the FCB, and the Task File Table (TFT). The operating system performs all input/output in terms of linknames; therefore, a linkname must be included on the FCB for a file. At execution, the linkname becomes a parameter to the executing program.

For discussion on the uses of linknames in specific operating environments, see VMOS Programmer's Reference Manual.

File Security

When a user creates a file, he is recognized as its owner. No other user can obtain access to the file unless the owner catalogs it with the SHARE=YES parameter of the CATALOG command or the CATAL macro.

File access to both the owner and to the sharer is also controlled by password options. Two passwords can be associated with the file: a read password and a write password, thus facilitating two levels of file sharing. The ACCESS parameter of the CATALOG command or the CATAL macro may be used to limit the file read access.

A file, marked as sharable, can be accessed by any user who can provide the identification code of the owner, the filename, and the appropriate password (if required).

If a file is being read by one or more tasks, and a subsequent task attempts to open the file for output (for example, UPDATE, OUTPUT), the system rejects the OPEN request. For ISAM files opened SHARUPD=YES, concurrent reading and writing of the same file is allowed. Competing tasks are essentially locked out at the PAM page level. If an attempt is made to read or write a file (that is, non-ISAM or SHARUPD=NO) currently opened for output, the system refuses to honor the OPEN request. The requesting program may choose to process another file or perform some other useful work.

Examples

User DOE catalogs the following files:

Example 1. CATALOG F1,SHARE=NO,ACCESS=READ
Example 2. CATALOG F2,SHARE=YES,ACCESS=READ
Example 3. CATALOG F3,SHARE=YES,WRPASS=C'0007'

Example 1. User SMITH cannot access F1 since DOE specified that the file cannot be shared. Note that even DOE, the owner, cannot write to the file.

Example 2. Smith, and in fact all other users, can read (but not write to) F2. Smith specifies the file as $DOE.F2; no password is required.

Example 3. Smith, and in fact all other users, can read and/or write F3 by specifying the name $DOE.F3 and the password 0007. Note that the default value for the ACCESS operand is WRITE.

Passwords required to gain access to protected files may be supplied in the PASSWORD command; they can also be specified in the file control block (FCB) which will be used to OPEN the file.

Password Requirements

Table 2-7 defines the password requirements for pertinent file management macros and commands. Note the following important points:

1. If a file has both a READ and a WRITE password, the WRITE password may be used to satisfy the requirements for the READ password; that is, the WRITE password allows reading and/or writing to a file.

2. If the file has no WRITE password, but has a READ password and if the chart specifies a WRITE password, then the READ password must be supplied.

TABLE 2-7. PASSWORD REQUIREMENTS

Command (or Macro)	Password Required: READ	Password Required: WRITE	Comments
OPEN INPUT	X		
OPEN REVERSE	X		
OPEN other types		X	
CATALOG		X	Update mode only.
COPY	X	X	READ password for input file; WRITE password for output file.
ERASE		X	
FILE		X	Required for a previously cataloged file if the SPACE parameter is specified.
FSTATUS			None required.

System Controller and File Security

The system controller can access any file in the system: that is, he can employ any file management command (macro) or access method consistent with the properties of the file. He has the same accessibility as owner of a file – but not more. For example, the controller must supply appropriate password (except when issuing the FSTATUS command or macro) to access the file. He cannot write to a file which is defined as read only (recall that neither can the owner). In other words, the controller is viewed as a coowner of every file.

File Retrieval

When a file is initially accessed, the system searches the owner's (the user who is logged-on) portion of the catalog for the specified file unless a $userid prefix is specified with the filename. In the latter case, the portion of the catalog of the user with the specified id is searched.

Typically, VMOS system files are cataloged under the system controller's id. To access the system's assembler, for example, the user would specify

$Controller-id.ASSEMBLER;

Since this notation is a bit verbose, the special id of null may be used to designate the system controller files. Accordingly, the user could write $ASSEMBLER to access the assembler.

Example

User Catalog DOE	User Catalog SMITH	Controller's (system) Catalog
PAYROLL	INVENTORY	ASSEMBLER
FORTRAN	PAYROLL	FORTRAN
ALGOL		ACCOUNT

File Referenced by DOE	Action
PAYROLL	DOE's PAYROLL file accessed.
$SMITH.PAYROLL	Smith's PAYROLL file accessed.
FORTRAN	DOE's "private" FORTRAN file accessed.
$FORTRAN	System's FORTRAN compiler accessed.
$Controller-id.ASSEMBLER	System's assembler accessed.
$ASSEMBLER	System's assembler accessed.
$SMITH.ACCOUNT	Error. Smith has no file called ACCOUNT.
$ALGOL	Error. The file is not in the system controller's catalog.

Programming Note

If the controller's file contains more than one simple name, the $Controller-id prefix must be used. For example, File A.B., if referenced as $A.B., would imply file B of user A.

Catalog Block Structure

When joined to the system, each user (including system controller) is assigned a primary catalog block of 2048 bytes, and the identification number of the block is stored in the system's join table. This block is the first block within the catalog file to which catalog entries (filenames) for the user are written.

Only catalog entries for a single user are contained in a given block. However, if the size of his catalog necessitates it, a user can own more than one block. All entries within a block refer to valid files and contain all the required information about the file.

The blocks within the catalog file are structured as shown in figure 2-6.

Entries are made in a catalog block on a first-come, first-served basis from left to right. If an entry is removed, the block's space counter is adjusted accordingly. If an entry other than the current last entry is removed, the entries are left justified.

When an entry cannot be contained in a block, a new block is obtained for this user. A block that has a successor contains the next block number in its control field. If an entry for other than the first block is removed, causing this block to become empty, the block is automatically returned to the system for reassignment. Entries are never moved from block i+1 to block i when an entry is removed from block i.

FIELD	①	②	③	④	⑤	6	.....	N
NAME	TOTAL NO. OF UNUSED BYTES IN BLOCK	USERID	BINARY NO. OF SUCCESSOR BLOCK		ENTRY 1	ENTRY 2		ENTRY M
SIZE	2	8	2	4	VARIABLE	VARIABLE		VARIABLE

FIELD DESCRIPTIONS:

① 2-BYTE BINARY COUNT INDICATING TOTAL NUMBER OF UNUSED BYTES IN THIS BLOCK. THE INITIAL VALUE OF THIS FIELD IS 2032.

② 8-BYTE USER IDENTIFICATION CODE.

③ BINARY NUMBER INDICATING THE LOGICAL BLOCK NUMBER OF THE BLOCK WITHIN THE CATALOG FILE WHICH SUCCEEDS THE CURRENT BLOCK OF THIS USER. IF NO SUCCESSOR BLOCK EXISTS, THIS FIELD IS 0.

④ NOT USED; RESERVED FOR FUTURE USE.

⑤ CATALOG ENTRY WHICH DESCRIBES A USER FILE.

FIGURE 2-6. CATALOG FILE BLOCK STRUCTURE

Example

Assume that a user block can contain four entries. The user performs the following operations:

USER OPERATION	CATALOG STRUCTURE			
CATALOG A.B	A.B			
CATALOG X	A.B	X		
CATALOG Y	A.B	X	Y	
ERASE X	A.B	Y		
CATALOG D	A.B	Y	D	
CATALOG E.Y.Z	A.B	Y	D	E.Y.Z
CATALOG R	A.B	Y	D	E.Y.Z
	R			
ERASE Y	A.B	D	E.Y.Z	
	R			
CATALOG T	A.B	D	E.Y.Z	T
	R			
ERASE R	A.B	D	E.Y.Z	T

FILE CREATION

The facilities supplied by VMOS support three methods of file creation: the use of software-supplied routines, card input in the background mode, and the operation of the user-written program. Each of these methods imposes its own conditions and requirements on the programmer, either as a result of routine operation or because of information the user's program is expected to contain.

Software-Supplied Routines

Software-supplied routines supplied with VMOS enable the programmer to create data files, source program files, and object program files while operating in the conversational mode. The use of these routines permits a programmer to circumvent much of the work associated with file preparation for processing. Most required, usually, is to specify the access method to be applied to the file. Open and close processing are accomplished by the routine.

Data files created by text editing routines can be used as input to user-written programs. The user is then required to supply correct open processing and access method application information. Open processing and access method application are discussed later in this section.

The several language processing and text editing routines are described in the VMOS Programmer's Reference Manual.

Background Card Input

The Command Language command, DATA, enables the programmer to catalog a file on a direct-access device and to insert data into it. If the file has been previously defined using the FILE or CATALOG commands, the DATA command can be used simply to insert data into the file. The DATA command can only be used to create SAM or ISAM files. The system assigns the following characteristics to the file: standard labels, variable length format records, and 2048-byte blocks. Labeling conventions are described in the VMOS Programmer's Reference Manual. Record formats are discussed in the Access Method portion of this section.

The programmer is not restricted to the creation of one file at a time when using card input. More than one file may be created during a card reading session using multiple DATA commands. The first DATA command must follow a LOGON command (see Task and Program Management – Section 2, Part 2). Each subsequent DATA command may follow the data cards associated with the preceding DATA command, or an END command. An END command must follow the last data card associated with the last DATA command in the card stream. A LOGOFF command must follow the last END command.

The DATA command may be used to create a file to contain any information from source language statements through raw data. However, VMOS considers the contents of the file to be data. The operating system does not differentiate among source language statements, data, and procedures. The contents of the file are stored but not processed when read in. The programmer must initiate processing during a different programming session.

The processing and creation of files by a user-written program require that the programmer supply the information necessary to identify the file, the format of its contents, and the way it is to be managed.

The basic set of instructional facilities supplied by the VMOS Command Language for identifying files and specifying the manner in which they are to be processed comprise the following: the FILE command (macro); the FCB (IDFCB) macro, OPEN macro, and CLOSE macro; and the macros associated with the various access methods.

File Identification and Operation of FILE Command

The FILE command (or macro) provides the facility to place a file's name in a catalog, allocate space for the file, assign devices to process the file, and complete or modify the File Control Block (FCB) – (see VMOS General Service File Management Macros discussion in the appendices). This command provides the link between user's program and the system, and provides the control program with information on how the user-written program intends to create or process the file.

Input and output files to be processed by a program must be identified to the system when the program is executed. The programmer may use the FILE command or its corresponding macro for this purpose. A FILE command may be entered conversationally or as part of a background initiated task; the FILE macro can only be used in the background mode and must be contained as part of the user-written program.

The FILE command provides a logical link between the user-written program and the files it is to process. For all files (input and output), the FILE command (macro) causes an entry to be placed in the Task File Table (see FCB discussion) and completes the File Control Block (FCB) if the file has not been previously cataloged. The FILE command (macro) also creates an entry in the catalog and provides for the allocation of space to contain the file. The FILE command (macro) is described in detail in Part 3, Section 2.

FCB Definition and Formation

In VMOS, the File Control Block (FCB) is the principal area of communication for all input/output operations associated with cataloged files. The FCB is the repository for all the information a program needs about a file in order to use the file. FCB contents depend upon the access method to be applied to the associated file and the type of processing (input/output) the file is to undergo. Regardless of the associated access method, FCBs exhibit common characteristics: same size for all access methods, and common fields within different FCBs are the same size and reside at same relative locations.

FCB SOURCE INFORMATION

Each file to be processed requires an FCB. The information necessary to construct it can be supplied from five different sources.

1. FCB macro instruction: When a program is assembled or compiled, an FCB may be created using the FCB macro. The FCB macro reserves space for a file control block, and may be used to supply information to the FCB and reserve space to contain the logical routines needed to process the file. Note that the logical routines for which the FCB reserves space are not considered part of the FCB. The FCB macro is discussed in detail under "VMOS General Service File Management Macros" in the Appendices. The FCB formats for various access methods are described in Part 3, Section 2.

2. User Program: The user-written program may complete or alter any or all fields of an FCB during program execution after the FCB has been completed but prior to the opening of the file.

3. FILE Command (or macro directive): Any fields, even if specified from sources 1 and 2 and for which the FILE command is a valid source, are completed at OPEN time. The FCB macro instruction parameters for each access method are summarized under the reference given in source 1 above.

4. User-program modification of the FCB when the file is opened through an open contingency exit using the OPEN parameter of the OPEN macro or File command.

5. The Catalog: Any fields not specified from the above sources, and for which the catalog entry is a valid source, are also completed when the file is opened.

FCB COMPLETION

The sequence of events significant to the processing of an FCB are summarized in chronological order below:

1. FCB macro: assembly (or compile) time.

2. User program object-time modifications prior to issuance of the OPEN macro.

3. OPEN time: FCB construction.

   a. FILE command (or macro)

   b. Catalog entry (regardless of OPEN mode)

   c. User program modifications at OPEN time through the contingency exit.

At this point, the FCB is considered complete. It is important to note that OPEN processing will not complete (or default) any unspecified fields. In other words, although OPEN processing accepts FCB parameters from a variety of sources, it does not validate, after having utilized these sources, that an FCB field is unspecified.

Open Processing and Function of OPEN Macro

Before applying an access method to a file, the programmer must request that an FCB be completed as a logical connection between the file and the problem program. The programmer issues an OPEN macro instruction that completes the FCB fields, verifies or creates file labels, positions volumes to the first record to be processed, and allocates buffer areas as required.

Additionally, the OPEN routine ensures that needed access routines are loaded and address relations are completed. The selection of access routines is governed by choices in file organization, buffering technique, input/output unit characteristics, and other factors.

In operation, some access routines are treated as part of the user's program and are entered directly rather than through a supervisor-call (SVC) interruption. These routines block and deblock records, control the buffers, and call the input/output supervisor when a request for data input or output is needed. Other routines, treated as part of the I/O supervisor and therefore executed in the privileged mode, perform error checks, prepare user-oriented completion codes, post interruptions, and bridge discontinuities in the storage areas assigned to a file.

OPEN also constructs a P2 (privileged) FCB, logically a protected extension of the FCB. This block contains a description of the extent (devices and boundaries) of the file. The privileged FCB is normally located via the FCB.

After operations on the file have been completed, the programmer logically disconnects the file from the problem program by issuing a CLOSE macro instruction. After an FCB has been closed, it may be reused. Furthermore, it is set to the same contents it had prior to issuance of the OPEN macro instruction.

The OPEN macro is discussed in detail under "VMOS General Service File Management Macros" in the appendices. Open processing options that are available are described in the Access Methods discussion of this section.

SEQUENCE OF EVENTS IN OPEN PROCESSING

The user's Catalog entry for a file contains more information that can be created or changed by File Management commands. This additional information is gathered from various sources, such as: the user's FILE command; the FCB; File labels; and the chain of elements called the Task File Table (TFT).

The user has the option to create an incomplete FCB in his program and/or to modify his FCB prior to Opening a file. If an incomplete FCB refers to an input file that is already cataloged and a FILE command is not presented to the system, the FCB is completed from existing catalog entries. Input label and retention period information is always taken directly from the catalog.

Foreign input tape label information is taken from the label itself, if the labels are standard and from the FILE command if nonstandard, and placed into the catalog prior to FCB completion. For output files, the information (traits) is taken from the FCB and placed into the catalog.

The central chain of elements that reflects all files and devices in use by a task is the TFT. In cases where the TFT was not created by the user's OPEN macro (that is, the referenced file is in HOLD status), the FCB is updated directly with the information in the existing TFT for that file.

The detailed sequence of events in OPEN processing is described below:

1. Search TFT entries for current linkname. (See also FILE Command and OPEN relationship under the OPEN macro.)

If linkname is not found, build TFT entry.

If linkname is found, update the user's FCB from the TFT (note, the TFT contains the FCB information presented by a previous FILE command). Remember, a linkname of spaces (binary zeros) will be treated as a special case. (See FCB explanation.)

2. Read catalog entry for the specified file. Complete unspecified fields in the FCB from the catalog, if possible.

Note: This function is not performed for a foreign tape file.

3. Mount necessary private volumes.

4. Branch to OPENX contingency exit.

5. Move traits from FCB to the catalog entry and then enter user's OPENZ contingency exit of the EXLST macro. (These functions are performed only if the OPEN type is OUTIN or OUTPUT.)

6. Validate passwords (not performed for foreign tape file at this time).

7. Allocate I/O buffers and validate buffer addresses, if necessary.

8. Process tape labels.

For foreign tape files, the following is now performed:

a. Move traits from catalog entry to FCB.

b. User program modifications through the OPENX contingency exit.

c. Validate passwords.

Notes:

1. Since the OPENX contingency exit for foreign tape files is taken relatively late in OPEN processing, certain fields within the FCB may not be modified.

2. Foreign tape files cannot be opened OUTIN or OUTPUT.

9. Access method OPEN processing constructs logical file processing routines if applicable.

10. Update catalog entry if foreign tape file is being processed, or if the file is being opened OUTPUT or OUTIN.

> CAUTION: Technically speaking, a file does not exist in the sense of containing data until the file has been Opened OUTPUT or OUTIN, and CLOSED. At CLOSE time, the file's creation date and expiration date (and last half page pointer for ISAM, PAM, and SAM) are inserted into the catalog.

FILE COMMAND (MACRO) AND OPEN MACRO RELATIONSHIP

Although an FCB must be associated with each file to be processed, another unit of control information is used to process files. This unit of information is called the File Definition Entry and includes the linkname. Although its creation and usage are not explicitly affected by the user program, a basic understanding of this entry is desirable.

The linkname (or logical filename) is a file-reference variable usually declared in the user program. It provides the link between the user's FILE command, the file itself, the FCB, and the TFT. It may be assigned references to files or devices and becomes a parameter to the executing program; although, strictly speaking, VMOS is file-oriented, not device-oriented. Some linknames belong to the system and are preinitialized, such as COBLIB (for the COBOL Source Library) and ERRFIL (for the post compilation diagnostic file).

DMS performs all file I/O in terms of linknames and the LINK parameter must be included in the FCB for a file. When this parameter is not specified, a default value of spaces is assumed.

The linkname also supplies the link for qualified filenames. For instance, COBOL conventions do not allow the use of qualified filenames, as the period has special meanings for COBOL. Thus, a COBOL program wishing to read a file called PAYROLL.A would identify a file called INPUTFIL and the user would supply the FILE command as /FILE PAYROLL.A, LINK=INPUTFIL, establishing the link between the two names.

The RELEASE command removes the association of a linkname with a file (or device). If the linkname reference was to a device, it will be relinquished (unless there was a KEEP parameter).

When a File Definition Entry is created, it is placed in a table called the Task File Table (TFT). The TFT is a chain of elements that is built by the OPEN (and the FILE or HOLD command) and reflects all files and devices in use by a task. The first element of the TFT is pointed to by the Task Control Block (TCB). Each element is identified as to creator, active or inactive, type (TOS or VMOS), and the length of the volume element. The DSECT IDTFT is used to reference the fields in the TFT. Figure 2-7 illustrates the relationship between the various commands, linknames, and TFT.

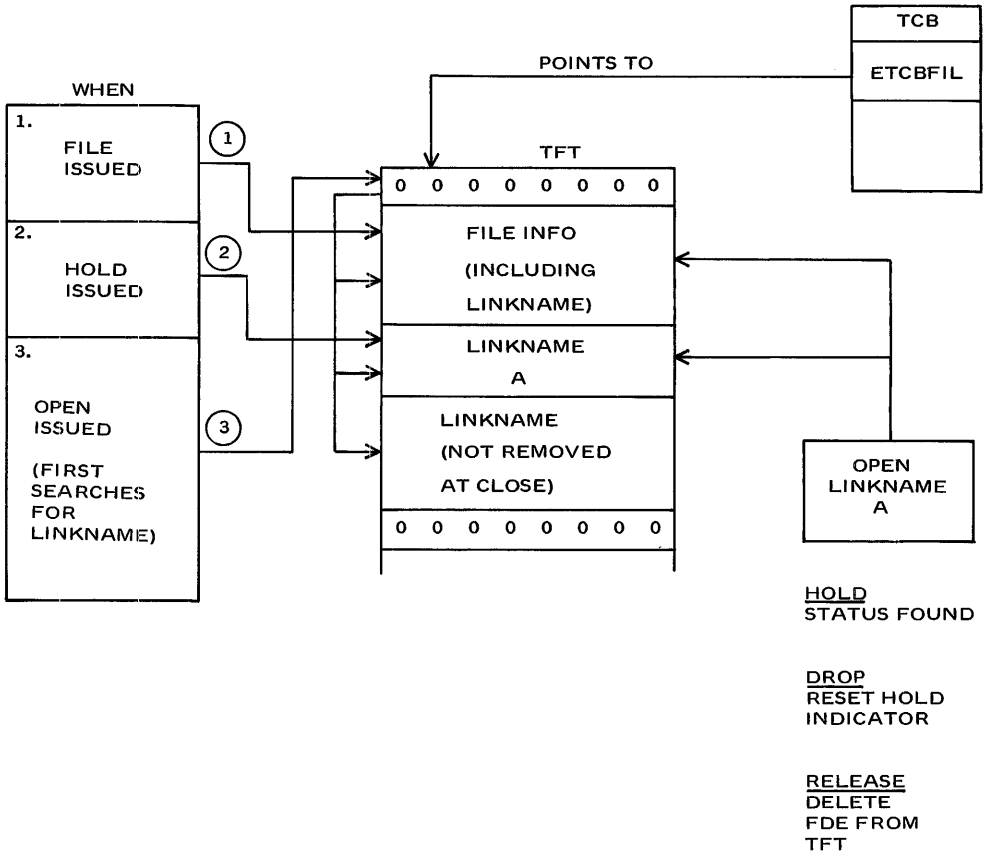


FIGURE 2-7. FILE DEFINITION ENTRY, LINKS, AND TASK FILE TABLE

The File Definition Entry can be created in one of three ways:

1. A FILE command (macro) is issued. Note that a FILE command is mandatory for a new file. All pertinent information specified in a FILE command is encoded and placed in this entry. Most of this encoded information is subsequently used by OPEN. The entry contains the symbolic linkname which makes the connection to other commands (for example, RELEASE) as well as to the FCB.

2. A HOLD command is issued for a file definition which has not yet been created. This is a trivial case, since nothing more than a skeleton entry is in the TFT.

3. An OPEN macro instruction is issued. OPEN searches the TFT for an entry with the linkname specified in the FCB. If one is found, information in that TFT entry is used by OPEN to complete its processing (for example, to complete and/or change FCB parameters).

If no TFT entry is found, then, OPEN creates one. If a TFT entry is created by OPEN, file characteristics are not moved from the associated FCB to the TFT. Note that when the file is CLOSED, this entry is not removed. Similarly, for example, a HOLD or RELEASE command could be issued for this TFT entry by specifying the linkname.

When a specified file, not prefixed by $userid, is to be Opened and is not found in the user's catalog, the controller's catalog is automatically searched. If such a file is found in the controller's catalog, and if the file is marked sharable, that file will be processed. Of course, password and access restrictions will still be obeyed.

> CAUTION: The automatic searching is performed by OPEN but not by the DMS commands of CATALOG, ERASE, FILE, and FSTATUS, although the FILE command passes on the filename to the FCB.

CLOSE Processing and Function of CLOSE Macro

Any file that has been Opened must be Closed to indicate that file processing is complete. At program termination, all files not closed by the program are closed by the system.

CLOSE processing consists of the following functions:

1. Releasing buffer areas automatically attained by the system and any other system-acquired control areas.

2. Performing all label verification and creation as specified by the label type. Refer to "Peripheral Conversion Routine" of the VMOS Service Routines Manual for complete details.

3. Updating the catalog entry if necessary.

4. Ensuring that pending write operations are completed.

5. Performing volume positioning as indicated in the CLOSE macro instruction.

6. Restoring the FCB to its original state (that is, as it was prior to the issuance of the OPEN macro-instruction).

7. Unlocking any data block the user has locked in closing the file.

The CLOSE macro is described in detail under "VMOS General Service File Management Macros" in the appendices of this manual.

Access Methods

The Access Methods supported in VMOS by its Data Management System allow the programmer to define his data formats and the method used to access this data. The user combines the FCB, OPEN, and CLOSE macros with a particular access method and the action macros associated with the performance of data handling operations. Table 2-8 summarizes the various access methods in terms of: record formats and device types associated with each, and the files created by other access methods which may also be used as input.

TABLE 2-8. ACCESS METHODS, RECORD TYPES, AND DEVICE TYPES

Access Method	Record Formats	Device Types	Other Access Method Files Allowed as Input
PAM	Fixed	Direct-access Tape (single reel, standard blocks)	SAM ISAM
SAM	Fixed Variable Undefined	Direct-access Tape	PAM
ISAM	Fixed Variable	Direct-access	PAM
BTAM	Fixed Undefined	Tape	PAM SAM
EAM	Fixed	Direct-access	

ACCESS METHOD RELATIONSHIPS

The privileged version of Primitive (or Primary) Access Method (PAM), used by various access methods (as well as privileged users), is summarized in figure 2-8.

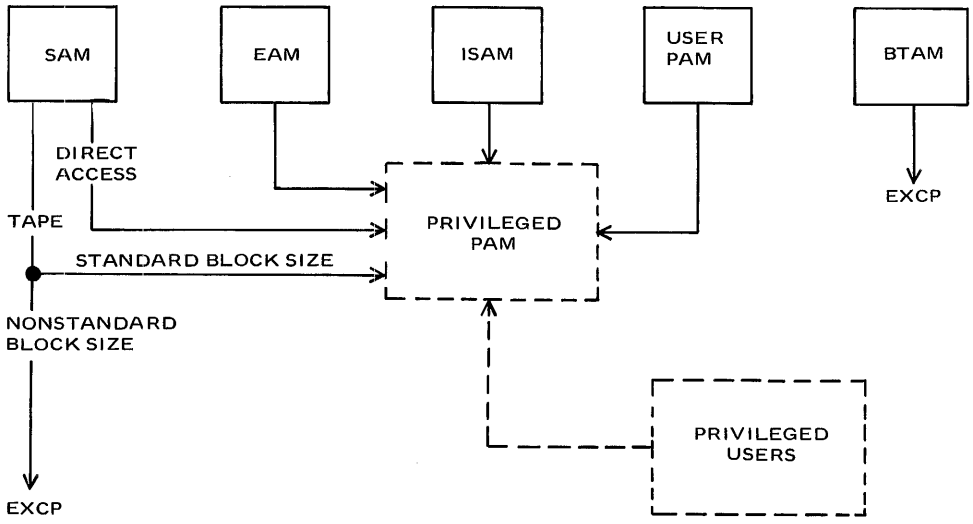


FIGURE 2-8. ACCESS METHOD RELATIONSHIPS TO PRIVILEGED PAM

Note that BTAM never uses PAM and that SAM (tape) need not use it. Accordingly, tape formats can be controlled as the user desires.

The most significant point concerning PAM is that all direct access volumes are assumed to be preformatted. PAM assumes – except for tape – that the file has been formatted into physical blocks of 2048 bytes (VMOS half-page or PAM Page), and that logically, each volume is partitioned into blocks of 2048 bytes. The number of such blocks will depend upon the device:

8564 (Disc Unit)

Each 8564 disc pack contains physical half-pages numbered 1 through 3046. The device is partitioned with each even numbered track containing one full half-page and the first half of another half-page. Each odd numbered track contains the second half of the split half-page from the preceding track and another full half-page. Each extent of a file must contain an even number of contiguous tracks on each cylinder (this does not imply that files have a cylinder orientation) of which the file is comprised. Furthermore, the disc drive must have the track overflow special feature.

8590 (Disc Unit)

Each 8590 disc pack contains physical half-pages numbered 1 through 12180. Each track contains three half-pages.

8568 (Mass Storage Unit)

Each magazine contains physical half-pages numbered 1 through 32768. Each track contains one half-page.

Magnetic Tape

Each reel contains n physical half-pages, limited only by reel length. Each half-page comprises one physical block on the tape.

In addition to data, each physical block of a PAM file contains a control field of 16 bytes structured as shown in table 2-9.

TABLE 2-9. PAM BLOCK CONTROL FIELD

coded file name	file version number	logical half-page number	Available to user if utilizing PAM directly. Access methods (ISAM, SAM, EAM) use this field for coded information.
No. of Bytes: 4	1	3	8

On direct access devices, this PAM block control field is recorded as the hardware key field. On magnetic tape devices, it is prefixed to the data, using the data chaining feature. Note that the key field makes the physical tape block 2064 bytes. Since the block is a multiple of 3, reverse processing is also available with 7-level tapes.

A block is defined as the unit of transfer to or from an I/O device. For example, it is the data between two gaps on a magnetic tape.

A standard block is a physical block (PAM page, VMOS half page) which consists of 2048 data bytes and 16 bytes of key. That is to say, it is the unit of transfer for PAM as described previously. Accordingly, any file (except a BTAM file) which is composed of standard blocks is always processed by PAM. All direct access files are composed of standard blocks.

A nonstandard block is a block of data (other than a PAM page) on magnetic tape which may be of any size less than or equal to 4096 bytes. There may be other size restrictions appropriate to the kind of tape drive employed (for example, a minimum block size of 12 bytes, a block size which is a multiple of 3 to allow reverse processing on 7-level tapes). Note that each nonstandard block in a file need not be the same size. No block can exceed the maximum size (specified in the BLKSIZE parameter in FCB).

Tape files processed by SAM may be composed of standard or nonstandard blocks. Of course, a given file cannot contain both kinds of blocks. When nonstandard block files are processed (either by SAM or BTAM), processing is effected by utilizing the physical level I/O macros (for example, EXCP) rather than PAM.

A buffer is defined as a contiguous area of memory. It is a portion of main storage into which data is read or from which it is written. If a block is nonstandard, then block and buffer are essentially equivalent. If a block is standard, the buffer must be a multiple of block size (that is, 2048, 4096,...) and may have a maximum value of 65,536.

Logical Records

The system supports three types of logical record formats:

1. Fixed length – format-F (all access methods).

2. Variable length – format-V (SAM, ISAM).

3. Undefined length – format-U (SAM, BTAM).

A fixed-length record is specified for files whose logical records all contain exactly the same number of bytes.

A variable-length record is specified for files containing logical records which vary in length and which contain user-supplied information that gives the length of the record. The first four bytes of each record is in the form llbb, where ll contains a binary number specifying the length of the record, in bytes; bb are two characters reserved for use by DMS and should not be utilized by the user. The bb portion of a record being added to a file may be modified by the access method processing logic during the action of moving the record into the file.

An undefined record is specified if the logical records in a file are neither format-F (fixed) nor format-V (variable). A format-U record is identical to a buffer. Accordingly, such records are a multiple of 2048 when standard blocks are specified.

It is important to note two points:

1. A logical record cannot exceed buffer size.

2. A logical record can exceed block size provided the file is composed of standard blocks.

The logical record types supported by the access methods are summarized in table 2-10.

TABLE 2-10. LOGICAL RECORD TYPES AND ACCESS METHODS

	Logical Record Type			
Access Method	F	V	U	Comment
PAM	X			Assumes 2048-byte block.
SAM	X	X	X	
ISAM	X	X		
BTAM	X	X*	X	Assumes one record per block.
EAM	X			Assumes 2048-byte block.

*Treated as undefined.

Examples of Record/Buffer/Block Relationships in SAM and ISAM

1. Assume:

Format-F records of 100 bytes
Standard blocks
Buffer size=2 blocks=4096 bytes.

The buffer is composed of 40 logical records; the rightmost 96 bytes of the buffer are unused; nevertheless, two 2048-byte blocks are written. Note that record 21 of the buffer is contained in two blocks; the first 48 bytes are in block 1, the last 52 bytes are in block 2.

2. Assume:

Format-F records of 100 bytes
Nonstandard block (tape only)
Buffer size=2020 (ostensible block size).

The buffer is composed of 2020 bytes; since only 20 records can fit into the buffer, a 2000-byte block is written.

3. Assume:

Format-F records of 1500 bytes
Standard blocks
Buffer size=1 block=2048 bytes.

This is a bad choice since 548 bytes are wasted. A desirable buffer size would be three blocks (6144 bytes). Now the buffer will contain four records, and 6000 bytes are utilized. Three blocks, each of 2048 bytes, are written.

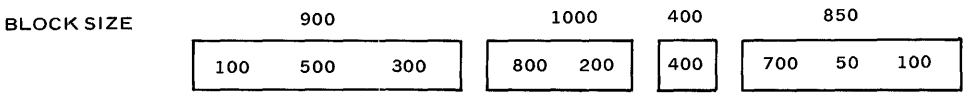
4. Assume:

Format-V records with the following sizes:

100, 500, 300, 800, 200, 400, 700, 50, 100

Nonstandard blocks (tape only) and BLKSIZE=1000.

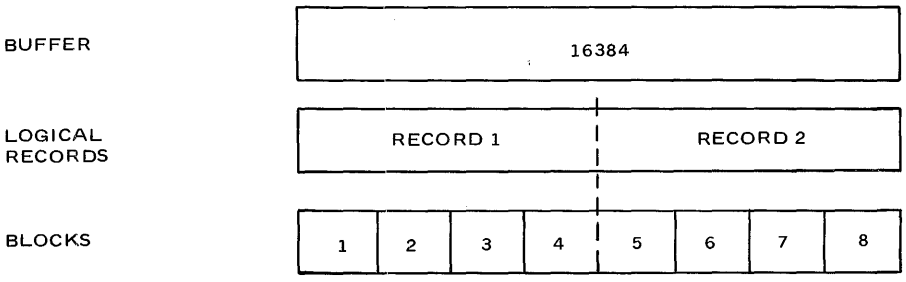
The following blocks are created:



5. Assume:

Format-F records of 8192 bytes
Standard blocks
Buffer size=8 blocks=1684 bytes.

Each buffer contains two logical records; each logical record requires four blocks.



Primitive Access Method (PAM)

The Primitive (or Primary) Access Method (PAM) provides the user with efficient means to access and create standard blocks in a random fashion. A block may be read from any portion of the file at any time. Similarly, blocks may be written to any portion of the file. PAM does not support logical record processing; blocking and deblocking must be accomplished by the user. Overlapped I/O operations are possible, if desired by the user.

The PAM macro instructions are as follows:

1. All general service macro instructions (for example, OPEN).

2. PAM which includes the following operations:

RD/RDWT – read data into core storage and optionally wait for I/O completion.

WRT/WRTWT – write data from core storage and optionally wait for I/O completion.

WT – ensure completion of an I/O operation.

CHK – determine completion status of an I/O operation.

PAM may be used to process any direct access file and any single reel tape file created with standard blocks. PAM may freely read SAM or ISAM direct-access files.

Logical file properties are placed in the FCB (for example, BLKSIZE, RECSIZE, RECFORM) so that the user program can perform appropriate functions (for example, deblocking). Regardless of the file's properties, PAM merely reads and/or writes one block at a time.

> CAUTION: In general, the user program cannot effectively process an ISAM file using PAM, since the complex relationship between ISAM logical records and indices is not apparent. However, PAM could be used to copy an ISAM file to tape (on a block-by-block basis). Similarly, a file created by PAM can be processed by other access methods. It is strictly the user's responsibility to construct a file in the exact format required by the access method (for example, no unused or missing blocks if processed by SAM; proper construction of the key field; construction of ISAM indices). Again, it is emphasized that such esoteric applications are beyond the requirements of most users.

PAM supports one record format:

Fixed length – format-F

Although short records may be read or written on direct-access volumes, the physical data blocks are always 2048 bytes in length. PAM assumes that there is exactly one record per block (or buffer, which for PAM is equivalent).

Opening a PAM File:

The file may be opened in the following ways:

1. INPUT: Retrieve records from an existing file.

2. OUTIN: Create a new file and retrieve records from the file. Note that labels are created since a new file is being generated.

3. INOUT: Retrieve records from an existing file and add and/or replace records. Note that labels are not created since the file is assumed to exist.

Table 2-11 defines the macro options which may be used with each type of OPEN.

TABLE 2-11. PAM MACRO OPTIONS VERSUS OPEN TYPE

PAM Macro Options	INPUT	OUTIN	INOUT
RD	X	X	X
RDWT	X	X	X
WRT		X	X
WRTWT		X	X
CHK	X	X	X
WT	X	X	X

The PAM macro is described in detail in Section 2 of Part 3.

Sequential Access Method (SAM)

The Sequential Access Method (SAM) provides a means for accessing records sequentially beginning at a specified point. This organization is most useful for making a sweep through the records of a file, including (for direct-access files) getting a record, updating it, and returning it to the file. Records can also be added to the file. Unlike ISAM, no additional space is required other than that needed by the logical records.

SAM automatically performs all blocking, deblocking, and buffering for the user. If the user requests the system to utilize only one I/O area (buffer), no buffering (overlap) can be performed.

Logical records are retrieved by use of the GET macro instruction. SAM anticipates the need for records based on their sequential order (the order in which they are written) and normally will have the desired record in storage, ready for use. Logical records are designated for output by use of the PUT macro. The program can continue as if the data record was written immediately, although the access method's routines may perform blocking with other logical records and delay the actual writing until the output buffer has been filled. Buffers are automatically scheduled by the system. SAM is, for the most part, device independent and allows files to be processed on magnetic tape and direct-access devices.

Transmission of data to and from the file may employ either of two modes:

Move mode. – The user specifies the location of the record in his program, and the system is responsible for transferring it to or from the buffers.

Locate mode. – The user requests the location in the buffer area of the current record. The user is responsible for transferring data to or from the buffers.

The following action macros are available in SAM to control file processing:

GET	Retrieves the next record from the file in physically sequential order.
PUT	Places a logical record in the file.
RELSE	Causes any remaining logical records in a buffer to be bypassed for input. For output, the next logical record created is written as the first record of a new buffer.
SETL	Specifies the position from which subsequent file processing is to take place.
FEOV	Advances the system to the next (tape) volume of a file before the end of the current volume is reached.
PUTX	Returns an updated logical record to the file (direct-access volumes only).

For files opened OUTPUT or EXTEND, SAM interprets each PUT or SETL macro instruction as an end-of-file indicator. The last PUT or SETL prior to CLOSE for a file thus automatically defines an end-of-file indicator for the system. If the user wishes to delete all records beyond a given record, he can use the SETL macro instruction to position the desired file point, and then issue a CLOSE macro instruction.

For files created with standard blocks, a retrieval address is made available to the user. The format of this address is described in detail at the end of this macro instruction under FCB retrieval address. When a logical record is stored (by execution of a PUT macro instruction), its retrieval address is made available in the FCB. The user can, if desired, construct another file from this retrieval address data and thereby establish a basis for subsequent nonsequential processing of the original file being created. Note that this retrieval address is also available in the FCB after the execution of the GET macro instruction. Accordingly, if the user did not create the file entered into the system, he can still create a secondary file from the GET retrieval addresses to facilitate nonsequential processing of the original file.

SAM RECORD FORMATS

SAM supports three record formats, as follows:

1. Fixed length – format-F

2. Variable length – format-V

3. Undefined length – format-U

> CAUTION: For format-U, SAM places one logical record per physical block (buffer). The user who specifies standard blocks (2048 bytes) and outputs 48-byte logical records would waste 2000 bytes.

The size of logical record cannot exceed the buffer size (refer to BLKSIZE parameter).

For OUTPUT files, SAM requires initial space allocation of at least one buffer length, plus an additional one buffer length if the user wishes to process in move mode.

Note that a record format other than that used to create the file can be specified when an existing file is opened INPUT or REVERSE. For example, a file which was created as a set of format-F records can subsequently be retrieved as a set of format-U records. Thus, the system would, in effect, return to the problem program a set of logical records (namely, all those in a buffer) for each GET macro instruction issued.

The following rules apply to 7-level magnetic tapes:

1. The physical block size (reference BLKSIZE parameter) must be a multiple of 3 if reverse processing is anticipated. It is not necessary that format-V or format-F logical records be a multiple of 3.

2. Format-U records must be a multiple of 3.

The file may be opened in the following ways:

INPUT — Retrieve records from an existing file in forward direction.

REVERSE — Retrieve records from an existing file in reverse direction. Multivolume tape files cannot be opened REVERSE.

OUTPUT — Create a new file or replace an existing file.

EXTEND — Add records to the end of an existing file.

UPDATE — Retrieve and replace records in an existing file. The PUTX macro is used in locate mode to rewrite logical records. Each record, however, must be first retrieved by a GET macro using locate mode. The user may not change the length of the record involved. This option is restricted to files on direct access devices.

Table 2-12 defines the action macros which may be used with each type of OPEN.

TABLE 2-12. SAM ACTION MACROS VERSUS OPEN TYPE

OPEN	INPUT	OUTPUT	EXTEND	UPDATE	REVERSE
GET	X			X	X
PUT		X	X		
PUTX				X	
SETL	X	X	X	X	X
RELSE	X	X	X	X	X
FEOV	X	X	X		

Other action macros maintain the retrieval address as follows:

GET — If the specified record causes a new buffer to be retrieved, bbbbbb is set to designate the buffer number for the record, and rr is reset to 00.

PUT — If the specified record causes the existing buffer to be written, bbbbbb is set to designate the buffer number for the record, and rr is reset to 00. That is, it is set to the number of the buffer in which the record will be placed.

RELSE — If the file is opened OUTPUT or EXTEND, bbbbbb is set to the number of the buffer in which the next record will be placed, and rr is set to 00.

FEOV — For tape, the retrieval address field is set to 00000100.

This field is only supported for tape files which are created with standard blocks.

It is important to note that bbbbbb is buffer oriented.

Example:

Assume BLKSIZE= (STD,2)

RECFORM= F

RECSIZE= 512

The retrieval address for the 10th record is 00000202; and 00000304 for 20th record.

Note that the system never increments the rr field beyond a single buffer. It is reset, as previously described, to zero when a new buffer is to be processed. The user can increment the rr field, if preferred, to maintain retrieval address information for each logical record.

USE OF PAM KEY IN SAM

The format of the rightmost eight bytes of the PAM Key in SAM files is: BBBXLLXX, where BBB is the buffer number in binary, X is unused and LL is the length of useful data in the buffer. Note that the first X byte (byte 4) is reserved for future system use.

Indexed Sequential Access Method (ISAM)

The Indexed Sequential Access Method (ISAM) processes logical records in an indexed-sequential file. It may be used to:

1. Create an indexed-sequential file in sequential or nonsequential manner.

2. Retrieve logical records of the file in sequential or nonsequential manner.

3. Update records in sequential or nonsequential manner.

4. Insert new records in proper logical sequence within the file.

5. Delete selected records from the file.

6. Retrieve the next sequential record in the file which contains requested flags.

Transmission of data to and from the files employs either of two modes:

Move mode. – The user specifies the location of the record in his program, and the system is responsible for transferring data to and from the buffers.

Locate mode. – The user requests the location in the buffer area of the current record, and is responsible for transferring data to or from the buffers.

The following action macros are available in ISAM to control file processing:

GET	Retrieves the next logical record in the file in ascending sequential order of the record keys.
GETR	Retrieves the next logical record in the file in descending sequential order of the record keys.
GETFL	Retrieves the next logical record satisfying the flag criteria.
GETKY	Retrieves the logical record with a specified record key.
PUT	Adds a logical record to the file. PUT is normally used to initially create the file by recording logical records in ascending sequential order according to their record keys.
PUTX	Replaces a record; retrieved by GET, GETR, GETFL, or GETKY.
INSRT	Places a new record in the file in the position determined by its record key.
STORE	Places a new record or replaces an old record in the file in the position determined by its record key.
ELIM	Eliminates a record from the file.
SETL	Specifies the location in the file from which subsequent processing is to take place. The beginning of the file, the end of the file, or the location of a record with the designated key may be specified.

ISAM RECORD FORMATS

ISAM supports two record formats, as follows:

1. Fixed length – format-F

2. Variable length – format-V

Only standard blocks are allowed; however, buffer size may range from 2048 to 65,536 bytes. The key must be contained within the record at a fixed position and be the same size for each record in the file.

OPENING AN ISAM FILE

When an indexed-sequential file is Opened, any of the following optional types of processing may be specified:

INPUT	Retrieves records from an existing file (sequentially and/or randomly).
OUTPUT	Creates a new file.
EXTEND	Adds records to the end of an existing file.
INOUT	Retrieves, deletes, replaces, and inserts records in an existing file. Additionally, records may be added to the end of the file.

OUTIN Retrieves, deletes, replaces, inserts, and stores records in a new file. Any information initially in the file is lost.

Table 2-13 summarizes the ISAM processing applicable to each type of OPEN.

TABLE 2-13. ISAM ACTION MACROS VERSUS OPEN TYPE

Action Macro	INPUT	OUTPUT	EXTEND	INOUT OUTIN
GET	B			B
BETR	B			B
GETFL	B			B
GETKY	B			B
PUT		B	B	M
PUTX				B
INSRT				M
STORE				M
ELIM				X
SETL	X			X

Legend:

M — action macro functions in move mode only.
B — action macro functions in move mode or locate mode.
X — action allowed.
A — blank entry specifies that the action is not allowed.

Note: Only the Move Mode is permitted for the PUT macro when the file is Opened INOUT or OUTIN.

Buffering (overlap) of I/O operations is supported for all modes of OPEN. Buffering is optional for the GET and GETR action macros.

Note that these macros do not require the supervisor call (SVC) instruction until all logical records in the current buffer have been processed.

If look-ahead buffering is not desired, the programmer should specifically omit the OVERLAP FCB parameter. When buffering is specified and the program switches from forward to reverse processing, or from random to sequential processing, the I/O to bring in the next sequential buffer is initiated immediately. Unless processing is strictly sequential, this can result in inefficient operation because of the system resources required to input data which is never accessed.

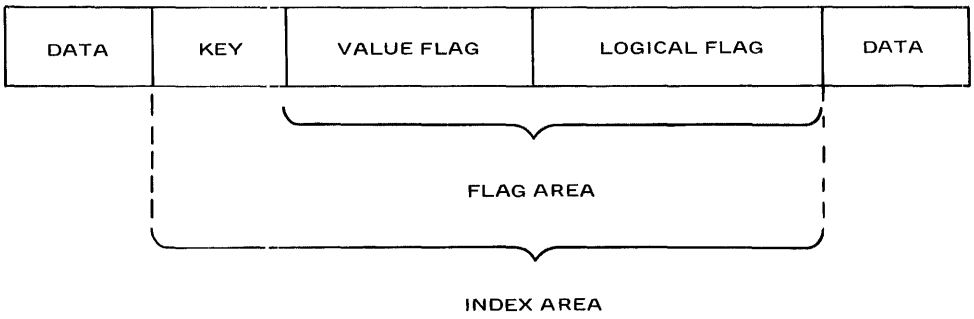
Note that PUT operations are always overlapped.

If padding factor is specified, space is reserved only during the sequential creation of the file via the PUT action macros. The INSERT and STORE action macros will attempt to use all the space available in a data block.

FLAGGED ISAM FILES

Flags are a method of limiting sequential searches on ISAM files. There are two types of flags: a value flag (such as tickler or date flag in TDOS ISAM) and a logical flag. Flagged ISAM files may contain value flags or logical flags, or both. The flag is part of the user's logical record; the value flag immediately follows the key and the logical flag immediately follows the value flag (or the key, if there is no value flag).

User Record:



The only constraints on key length, value length, and logical length is that their sum must not exceed 255 and the key length must be at least 1.

Flag Information is propogated upwards through all levels of the ISAM index.

Value Information is propagated by either a MAX or a MIN function as specified by the user. If the (MAX, MIN) function is specified, each Index entry will contain the (highest, lowest) value flag of the next lower level block (data or index) to which it points. The method of value flag propagation used is a function of the data in the file and how the file is normally going to be searched. Logical flags are propagated by OR'ing together all the flags of the next lower block. Note that this destroys zeros; therefore, if a file is to be searched for males on one occasion and females on another, two bits are required to indicate sex.

Flags limit searching of ISAM files at the highest index level possible.

Opening a Flagged ISAM File:

If either VALLEN or LOGLEN is unequal to zero, the file is assumed to be flagged. If the file previously exists (OPEN type is INPUT, INOUT, or EXTEND), the catalog values of VALPROP, VALLEN, and LOGLEN may be defaulted, but they cannot be changed by the FCB macro or FILE command.

ISAM SHARED FILE UPDATE

The ISAM shared file update facility permits two or more users to open the same ISAM file simultaneously for the purpose of updating it. The facility is designed around two basic concepts:

The attribute of file sharability is defined by the user when the file is Opened.

To avoid conflicts between two or more programs in attempting to access and update the same data records, a locking and unlocking mechanism is available to the user at the data block level. Where lock/no lock options are superfluous and locking must be accomplished, ISAM provides automatic locking of the data block.

Shared update facilities available to the user are described below in detail.

OPENING AN ISAM SHARED FILE

The first user to open an ISAM file may open it with any combination of values for OPEN and SHARUPD. The following chart indicates (by an X which OPEN/SHARUPD combination will be permitted when user B tries to open the same file user A has already opened. If more than one user has already opened the file, the OPEN/SHARUPD combination of user B will be compared against the OPEN/SHARUPD combination of every user who has opened the file; user B can open the file only if each such comparison permits him to do so.

			USER B									
			SHARUPD=YES					SHARUPD=NO				
			INPUT	INOUT	EXTEND	OUTIN	OUTPUT	INPUT	INOUT	EXTEND	OUTIN	OUTPUT
USER A	SHARUPD=YES	INPUT	X	X	X			X				
		INOUT	X	X								
		EXTEND	X	X								
		OUTIN	X	X								
		OUTPUT	X	X								
	SHARUPD=NO	INPUT	X					X				
		INOUT										
		EXTEND										
		OUTIN										
		OUTPUT										

When more than one user is updating the same file, it is necessary to lock portions of the file briefly in order to prevent two users from attempting to update the same record simultaneously. This locking is done at the data block level. Some of the ISAM action macros (i.e., PUT, STORE, INSRT, and ELIM (KEY)) are "self-contained," that is, one macro locks, updates, and unlocks the necessary block. Others require two macros to perform the entire sequence: a GET, GETR, GETKY, or GETFL to lock the block and a PUTX or ELIM (no KEY) to update and unlock the block. As a consequence, an optional LOCK/NOLOCK parameter may be specified for a GET, GETR, GETKY, or GETFL macro.

If a data block has been locked by a GET, GETR, GETKY, or GETFL which specified LOCK, then other users can read the block (i.e., can refer to a record in the block with a SETL or with a GET, GETR, GETKY, or GETFL which specifies NO LOCK), but other users cannot update the block (i.e., cannot issue a PUT, STORE, INSRT, ELIM, or PUTX which will update the block) or lock it with a macro specifying LOCK. If NOLOCK is specified, then other users can read, lock, or update the block.

If SHARUPD = NO, then the LOCK parameter is ignored in the GET, GETR, GETKY, or GETFL macro processing and no locking occurs.

USE OF LOCK/NOLOCK PARAMETER WHEN SHARUPD=YES

A data block locked by one user can be read but not updated by other users, and it is not possible for two users to lock the same data block simultaneously. On the other hand, a user who is reading a data block but has not locked it permits other users to read or update the block. Thus, the use of NOLOCK in the GET, GETR, GETKY, and GETFL macros increases the degree of data block sharability. However, LOCK must be specified if the data block is to be updated via a PUTX or ELIM (no KEY) or if the user wants to ensure that no other user will update the data block while he is reading it.

Locking Data Blocks

User may have more than one ISAM file opened with SHARUPD = YES, but can only have one data block locked at a time.

Conceptually, the first step of every ISAM action macro which will lock a data block is to unlock any data block this user has locked in any file. Also, every ISAM action macro will unlock any locked data block in the file to which the macro is issued. (GET, GETR, and PUT will not actually unlock a data block if the same data block would be relocked later in the macro processing, and PUTX and ELIM (no KEY) do not unlock the data block until the updating is completed.)

Implicit Locking and Unlocking of Data Blocks

In addition to the explicit locking done by GET, GETR, GETKY, and GETFL and the explicit unlocking done by PUTX and ELIM (no KEY), each ISAM action macro can cause certain data blocks to be locked or unlocked.

If SHARUPD = YES, a PUT will lock the last data block in the file and leave it locked at the end of the macro. Also, if SHARUPD = YES, a STORE, INSRT, or ELIM (KEY) will lock the affected data block before the update and will unlock it when the updating has been completed.

In order to avoid a deadlock situation, each user can have at most one data block locked at one time, regardless of how many files he has opened. To ensure this, when a user issues an ISAM macro which will lock a data block (i.e., SHARUPD = YES for the file and the macro is a STORE, INSRT, ELIM (KEY), or PUT or is a GET, GETR, GETKY, or GETFL specifying LOCK), any data block that user has previously locked (in any of his files) will be unlocked. If a locked data block is in the file to which the macro refers, it will be unlocked even if the macro will not lock a data block (except that the data block will not be unlocked if the macro is a GET, GETR, or PUT which will relock the same page).

When a PUT is issued for a file opened in locate mode, IOREG is updated and the appropriate data block is locked. The user must move the record to or build the record at the location specified by IOREG before the data block is unlocked (i.e., before execution of another ISAM action macro).

Updating a Data Block

Before a data block can be updated, that block and all associated index pages must be locked.

If an index page is split during the update, one of the two resultant pages will be kept in the buffer at that level and one will be written out.

When the update is complete, all index pages and the data block are unlocked.

CONTINGENCY EXITS

LOCK Exit

Control is given to the user at this LOCK exit whenever he is unable to OPEN his file due to a conflict between his OPEN/SHARUPD specifications and those of the users who have already Opened the file.

USERERR Exit

In addition to other uses of the USERERR exit, if the file is opened with SHARUPD = YES, the user will be given control at this exit if he issues a PUTX or ELIM (no KEY) without first locking the data block or if he issues a GET, GETR, or GETFL after taking his PGLOCK exit without first repositioning his file (as described in the PGLOCK Section below).

PGLOCK Exit

This exit is meaningful only if the file was opened with SHARUPD = YES. Control will be given to the user at this exit whenever a data block (or index page), which must be referenced in the course of processing an action macro, is inaccessable because of the manner in which another user is accessing it. (For example, if one user has locked a data block and a second user tries to lock it, the second user will be given control at his PGLOCK exit.)

It is possible for any macro issued for a file opened with SHARUPD = YES to cause control to be passed to this exit. Unless caused by a PUTX or an ELIM (no KEY), the "internal pointer" will be invalid when the PGLOCK exit is taken. Thus, it is necessary to reposition this pointer before issuing a macro which assumes that it is valid (i.e., GET, GETR, and GETFL). The pointer can be repositioned by issuing the RETRY macro or one of the following ISAM action macros: GETKY, SETL, PUT, STORE, INSRT, or ELIM (KEY). If a GET, GETR, or GETRL is issued before the pointer is repositioned, control will be passed to the USERERR exit. If the macro which caused the PGLOCK exit to be taken was a PUTX or an ELIM (no KEY), the data block will still be locked at the PGLOCK exit and no repositioning is necessary.

CONSIDERATIONS IN USING SHARED UPDATE FACILITIES

When to Use ISAM Shared Update

Specifying SHARUPD = YES for a file causes detailed tables to be set up in class 4 memory, while SHARUPD = NO only causes the creation of one or possibly two table entries. In addition, the detailed tables created by SHARUPD = YES must be maintained at every ISAM macro, while the entry created for SHARUPD = NO is only updated at OPEN/CLOSE time. Thus, in order to save time and class 4 memory, SHARUPD = YES should be specified only when it is known that more than one user will be opening the file at the same time, and that at least one of the users will be updating it (i.e., will have OPEN = INOUT, OUTIN, OUTPUT, or EXTEND).

Opening and Closing an ISAM Shared Update File

When a file is opened with SHARUPD = YES, detailed tables are created in class 4 memory. These tables must be checked and maintained by all users of ISAM. In general, to avoid unnecessary overhead, a user should not open a file until ready to process it, and should close it as soon as its processing is finished.

CLOSE processing will unlock any data block the user has locked in the file being closed.

Locking and Unlocking Data Blocks

Issuing macros which alternately lock and unlock data blocks can cause unnecessary overhead. In some situations, this overhead is unavoidable, as in the case of the GET/PUTX sequence, where the GET must lock the data clock if SHARUPD = YES and the PUTX must unlock it. However, through careful program design, the user may be able to avoid such situations as issuing GETs which specify LOCK alternately to two or more files or issuing GETs which alternately specify LOCK and NOLOCK to the same file. This is especially important where GET, GETR, and PUT are concerned, because locking and unlocking require an SVC, while GET, GETR, and PUT can usually be processed by P1 code.

Use of Duplicate Keys with a Shared Update File

When a user is reading a file sequentially (i.e., is issuing GETs, GETRs, and GETFLs) and the file is opened with SHARUPD = YES, he must be careful if there are duplicate keys in the file. If another user had modified an index page associated with the data block this user is reading, it may be necessary for ISAM to reposition the user in the file based on the modified index pages. (in effect, ISAM issues a SETL KEY to reposition the user.) However, if the key involved is one of a sequence of duplicate keys, the user may be positioned at an unexpected record in the sequence. It is the user's responsibility to keep track of which records in the sequence he has processed.

The RETRY macro may cause the same unexpected positioning for a file with duplicate keys. This condition is described in detail under the section devoted to the RETRY macro.

File Reconstruction RESET Command

When more than one user opens an ISAM file with SHARUPD = YES and OPEN = INPUT, the File Reconstruction System notes only the first OPEN and the last CLOSE (ignoring any nested or overlapped OPENs and CLOSEs) and uses this outermost OPEN/CLOSE pair in the OPEN count. Using the COUNT parameter in a RESET command for such a file may give unexpected results (i.e., the file may be reset to an earlier OPEN than the user expects). Thus, for these files, the TIME parameter should be used instead of the COUNT parameter in requesting the resetting of the file.

ISAM FILE STRUCTURE

An ISAM file is made up of data blocks (2048 bytes) and index blocks. Entries in the data blocks contain the user's logical records. Entries in the index blocks contain pointers to either lower level index blocks or, in the case of the lowest level index block, to data blocks. Specifically, for each data block in the file, there exists at the lowest level an index entry of size KL + 4 where KL is key length [+LOGLEN+VALLEN].

The data blocks, which are also describable as level 0 blocks, have their records ordered by record key. Moreover, the records in data block i all have keys greater than or equal to those in data block i-1. Similarly, the records in data block i all have keys less than or equal to those in data block i + 1.

One should note that an ISAM file is logically structured as a multilevel tree, as illustrated in figure 2-9.

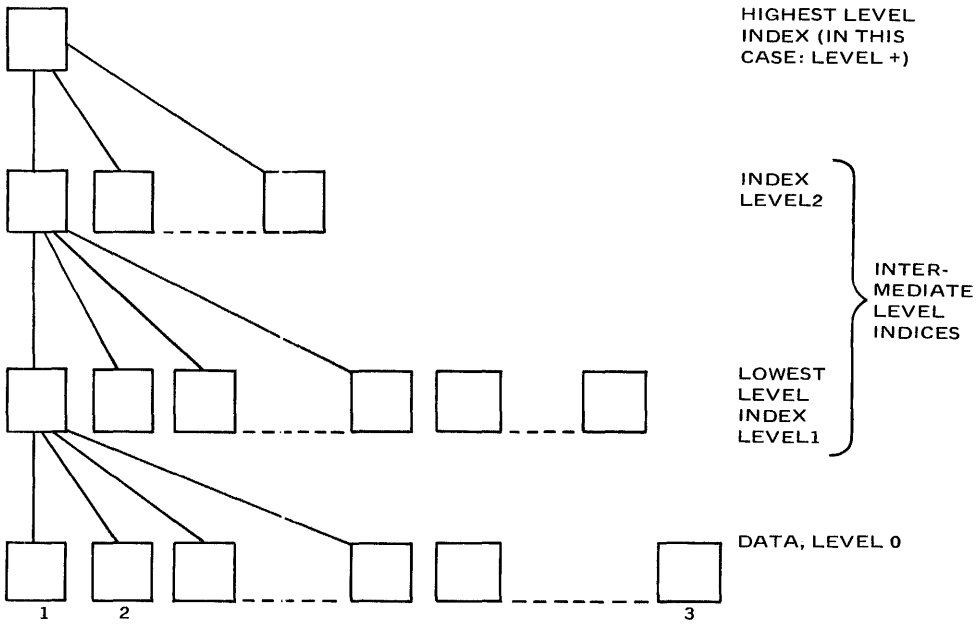


FIGURE 2-9. ISAM FILE INDEX/DATA STRUCTURE

The highest level index block is always present in the file (even if the file is null, i.e., contains no data blocks). In addition to containing pointers, this index block contains 36 bytes of ISAM label information which is used internally by ISAM. Therefore, in the calculations that follow, the size of the highest level index block is effectively 2012 bytes rather than 2048 bytes.

For modest sized files there are no intermediate level index blocks. The highest level block, being at level 1, would therefore contain pointers to data.

Whenever a file grows so large that the highest level index block does not have enough space to contain pointers to all of the blocks on the next lower level, that block is split in half and a new highest level index block is created, one level higher than the old and containing two pointers, one to each half of the split block.

Size of ISAM Files

Because of the many options which a user can exercise in the creation of an ISAM file, it is, in general, not simple to calculate the size of the file, the number of index blocks, or the number of levels of index. The following formulae are provided as approximations only.

Data Blocks:

RDB = FLOOR (BS/RS)

NDB = CEIL (NR/RDB)

where:

FLOOR = the greatest integer less than.

CEIL = the smallest integer greater than.

RDB = Records per Data Block.

BS = Block Size.

(a) Blocksize = 2048 times the number of half-pages specified in the BLKSIZE parameter.

(b) If the file is created sequentially by means of the PUT action macro, BLOCKSIZE must be adjusted by being multiplied by (100-PAD)/100 (note, the default value of PAD is 15).

(c) If the file is created either sequentially or randomly by means of the STORE or INSERT action macros, the file is (to the first approximation) densely packed. Therefore the value of BLOCKSIZE given in (a) above should be used regardless of any PAD specification.

RS = Record size.

(a) For F-type records (RECFORM=F), add 4 to the value specified in RECSIZE.

(b) For V-type records, use the average record length.

NDB = Number of Data Blocks.

NR = Number of Records.

Index Blocks:

$$NB_i = CEIL\left(\frac{NB_{i-1} * (KL+4)}{Size}\right)$$

where:

NB_i = Number of blocks at level i.

NB_i = Number of blocks at level i-1 (note NB_0 NDB).

KL = Key/flag length as specified as the sum of KEYLEN, LOGLEN, and VALLEN parameters.

Size = The effective size of the index block at level i.

(a) If level i is the highest level Size=2012 bytes, otherwise Size=2048 bytes.

(b) If the file is created sequentially, either by the PUT, STORE, or INSERT action macros, the Size given in (a) above must be adjusted by multiplying by .5 except when i is the highest level index.

(c) If the file is created completely randomly, the value given in (a) above is approximately correct.

(d) If the file is created with random sets of sequential PUTs, STOREs, or INSERTs, some adjustment factor between .5 and 1.0 should be used.

Examples:

Assume:

STD blocks
1000 80-byte records
KEYLEN = 8
LOGLEN = 2
VALLEN = 2
RECFORM = F
PAD = 10
File created sequentially with the PUT action macro.

RDB = FLOOR (2040*.9/(80+4)) = 21

Each data block therefore contains 21 records.

NDB = CEIL (1000/21) = 48

There are 48 data blocks in the file.

$$NB_i = CEIL \left(\frac{(48 * (8+4))}{2012}\right) = 1$$

Thus, only one index block is required; the total number of half-pages in the file is 49.

Assume:

STD blocks
100,000 100-byte records (average)
KEYLEN = 5
LOGLEN = 1
VALLEN = 0
RECFORM = V
File created sequentially with the STORE action macro.

RDB = FLOOR (2048/100) = 20

Each data block therefore contains approximately 20 records.

NDB = CEIL (100000/20) = 5000

there are approximately 5000 data blocks in the file.

$$NB_i = CEIL \quad \frac{(5000*(6+4))}{(2048*.5)} = 49$$

Thus there are approximately 49 level-1 index blocks in the file.

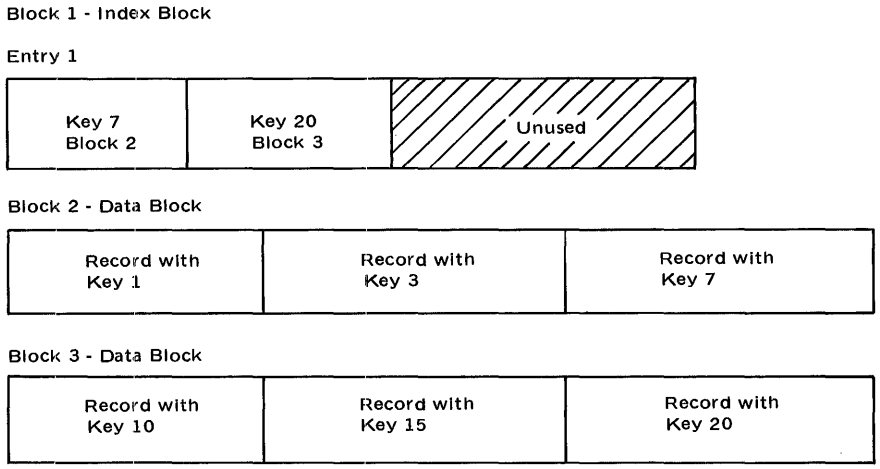
$$NB_2 = CEIL \quad \frac{(49*(6+4))}{2012} = 1$$

The highest level index block in the file is therefore level 2. It can be seen, from the above equation, that the file would have to contain more than 4 times the specified number of data records before a third level of index would be required. The total number of half-pages required by the file as originally specified would be approximately 5000+49+1=5050.

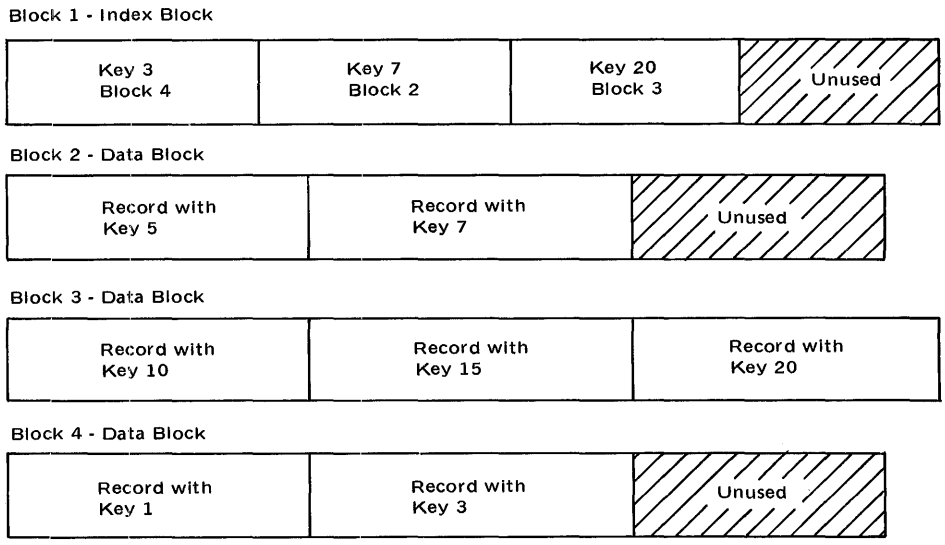
If a record does not fit where the index indicates it should be, the data block is "split" such that approximately one-half of the data records are moved from the original block into a new, empty block.

Example of Splitting Data Blocks

Assume the following file exists:



The user now specifies that a record with a key of 5 is to be inserted. Since no room exists in block 2, which is the logical block that the record should be retained in, block 2 is split and the file now has the following structure:



ISAM action macros (including OPEN) can be described in terms of two components:

1. Move an internal pointer to a specified record.

2. Perform the desired action on the record now pointed to.

It is important to note that:

a. The pointer is moved before the action is performed.

b. Certain macros may not require that the pointer be moved or may specify that only the pointer is to be affected and no other action be performed.

Table 2-14 summarizes ISAM macros in these regards.

TABLE 2-14. SUMMARY OF ISAM POINTER RULES FOR ACTION MACROS

ISAM Action Macro	Pointer	Action	Comments
OPEN	Positions the pointer to an imaginary record just before the first record in the file.		OPEN is listed here only to specify the pointer result. The action resulting from an OPEN is described elsewhere.
GET	Moves the pointer forward one record.	Retrieves the record pointed to.	1. If the pointer is moved beyond the current end of the file, the user is given control at his EOFADDR address. 2. If the previous macro was a SETL or ELIM to a record with a specified key, the pointer is not moved before the record is retrieved.
GETR	Moves the pointer backward one record.	Retrieves the record pointed to.	1. If the pointer is moved beyond the current beginning of the file, the user is given control at his EOFADDR address. 2. See GET, comment No. 2.
PUT	Moves the pointer to just beyond the current end of the file, if not already there.	Places the record in the file at the position pointed to.	
GETKY	Moves the pointer to record with the specified key, or to the position in the file where such a record would exist, if a matching key is not found.	Retrieves the record pointed to, if it exists.	For the sake of describing the pointer placement only, one can consider a GETKY of a non-existing record to be equivalent to a GETKY of a record of length 0 in between two existing records. Thus, a succeeding GET or GETR will work correctly.

(Continued)

TABLE 2-14. SUMMARY OF ISAM POINTER RULES FOR ACTION MACROS (Continued)

ISAM Action Macro	Pointer	Action	Comments
PUTX	Pointer not moved.	Places the record in the file at the position concurrently pointed to.	Note that additional checks are made to ensure that a successful GET, GETR, GETFL, or GETKY has been issued just prior to a PUTX. Thus, the fact that the pointer is not moved cannot cause inadvertent errors.
INSRT	Moves the pointer to the position specified by the record key.	Inserts the record at the location pointed to.	This record will not be inserted if a record already exists with the given key. However, the pointer will still point to the place in the file where the record should have gone. This record can be described as a record of 0 length — therefore a GET following an unsuccessful INSRT will retrieve the duplicate record.
STORE	Moves the pointer to the position specified by the record key.	Stores the record at the desired location.	If a record(s) with the given key already exists in the file, the pointer is set to the position immediately beyond the duplicate(s). Then the record is stored.
ELIM	If KEY is specified, moves the pointer to the (first) record with the given key. Otherwise the pointer is unmoved.	Eliminates the record from the file.	Note that ELIM can be described as (although it is not) a left-shift operation of that portion of the file to the right of the given record. Therefore, successive ELIM's need not (conceptually) move the pointer.
SETL	Move the pointer to 1. An imaginary record just prior to the first record in the file (if B is specified). 2. An imaginary record just after the last record in the file (if E is specified). 3. The (first) record with the given key, or the place in the file where such a record would exist.	No action takes place.	1. Use of a SETL effectively nullifies the pointer movement associated with the next succeeding GET or GETR. 2. For the purpose of clarifying the pointer placement for a successive GET or GETR, a SETL to a nonexistent record can be considered as a SETL to a record of 0 length in the correct position, with the pointer action of the next succeeding GET or GETR not nullified.
GETFL	Moves the pointer to the record retrieved. If no record is retrieved, the file is positioned such that a subsequent GET (GETR) will retrieve the record with a key greater than (less than) or equal to the key specified as a limit, or result in an End-of-File.	Retrieve the next sequential record which satisfies the flag criteria.	The record which would have been retrieved by a corresponding GET or GETR is the first record investigated for satisfying the flag criteria when a GETFL is issued.

The rightmost eight bytes of the PAM key in ISAM files is:

LLPPUUFF,

where:

LL is the page length (that is, the highest byte position used in the page).

PP is a pointer to the first logical record in a data page; if this is an index page, the value is 0.

UU is only meaningful while the page is in memory. It contains the location of the most recently used record.

FF is a pointer to the first sector of free space in the page. If there is no free space, FF points to the end of the page. If this is an index page, the value is 0.

Note that ISAM records within a data block are chained (linked); record i points to record i+1; the last record points to record 1.

Basic Tape Access Method (BTAM)

The Basic Tape Access Method (BTAM) provides the programmer with an efficient and flexible means for storing and retrieving the blocks of a sequentially organized tape file.

Macro instructions of BTAM may be used in a device-dependent manner, allowing the programmer to gain access to data in other than strictly sequential order. In addition, control of I/O devices is provided: for example, positioning tape volumes.

A major BTAM feature is that the program may read and/or write data blocks and change direction without requiring intervening CLOSE operations.

BTAM macro instructions are as follows:

All general service macro instructions (for example, OPEN).

BTAM – which includes the following operations:

RD/RDWT – read data into core storage and optionally wait for I/O completion.

REV/REVWT – read data reverse into core storage and optionally wait for I/O completion.

WRT/WRTWT – write data from core storage and optionally wait for I/O completion.

WT – ensure completion of an I/O operation.

CHK – determine completion status of an I/O operation.

A set of control codes to position tape volumes and write tape marks.

BTAM RECORD FORMATS

BTAM supports two record formats:

1. Fixed length – format-F

2. Undefined length – format-U (format V is treated as format U)

In either format, BTAM assumes there is exactly one record per physical tape block (or buffer, for which BTAM is equivalent).

Thus, in a real sense, these are not logical records. The format-F type merely conveys to BTAM that the block size specified in the FCB parameter BLKSIZE is to be used unless the count field is specified in the action macro instructions. The format-U type conveys that the block size is contained in the register specified in FCB parameter RECSIZE, unless the count field is specified in the action macro instructions. Note that the minimum length record that can be written to a magnetic tape is 12 bytes long.

An existing file, created by SAM, can be processed by BTAM. The user is cautioned that files created by SAM are his responsibility to deblock and properly process. This processing can be particularly erudite where the file, created by SAM, was created with buffers which were two or more standard blocks. The complexity arises since logical records may "straddle" physical blocks.

OPENING A BTAM FILE

The file may be opened in the following ways:

INPUT	Retrieves records from an existing file.
REVERSE	Same as INPUT except tape is positioned to the end of the file at OPEN time. Multivolume files cannot be opened reverse.
OUTPUT	Creates a new file.
INOUT	Retrieves records from an existing file and adds and/or replaces records. Note that header labels are not created since the file is assumed to exist.
OUTIN	Creates a new file and/or retrieves records from the file. Note that labels are created since a new file is being generated.
SINOUT	Same as INOUT except that the tape is not positioned (that is, OPEN in-place). This option is not allowed if the tape is at BT.

Table 2-15 defines the action macros which may be used with each type of OPEN.

TABLE 2-15. BTAM ACTION MACROS VERSUS OPEN TYPE

Operations	INPUT	REVERSE	OUTPUT	INOUT OUTIN SINOUT	Comments
RD|RDWT|REV|REVWT	X	X		X	
WRT|WRTWT			X	X	
WT	X	X	X	X	
CHK	X	X	X	X	
Control	See Comments	See Comments	X	X	Output Control functions are prohibited.

X = allowed

Programming Notes

The only difference between INPUT and REVERSE is that of positioning the file at OPEN time. Thus, an RD or RDWT operation, if a file is opened REVERSE, does not implicitly specify that a block is to be read in a reverse direction. Note that a file opened INPUT can be read in the forward and/or reverse direction.

Evanescent Access Method (EAM)

EAM is a specialized access method designed primarily to process temporary files in an optimum manner. Thus, the acronym may suggest the Efficiency Access Method or the Evanescent (temporary) Access Method.

Efficiency is achieved in the following ways:

EAM files are not cataloged.

VTOC operations relating to space assignment and label processing are not performed.

Opening of an EAM file requires zero disc pulls.

FILE card processing, FCB completion, and device assignment operations are not performed.

The counterpart to the above operations are not performed at CLOSE or ERASE time.

Fetching of data, with the action macros, utilizes less CPU time.

Processing areas (for example, FCB, logical routines) required for file operations are considerably smaller than those for the standard access methods (ISAM, SAM, etc.).

This efficiency is achieved by specializing and limiting the functions which EAM will support. Illustrative restrictions demonstrate this:

An EAM file is temporary.

It cannot be shared. In particular, the file cannot be multiply opened (that is, if a particular EAM file is open, another program (routine) cannot open it). Note that a task may concurrently process different EAM files.

No user storage allocation, device assignment, or label processing is permitted.

Users can only transfer 2048-byte blocks of data.

An EAM file always resides on a public volume(s).

An EAM file is not supported by the Checkpoint/Restart components of the system.

EAM SPACE UTILIZATION

The system controller for an installation defines a file named SYSEAM. This is a normal, cataloged PAM file. It is the file which EAM uses to retain the various EAM files in the system.

At the very first session, the SYSGEN process allocates 24 PAM pages for EAM files. When EAM requires additional space, it requests a block of 144 PAM pages each time from the system allocator. If the request is granted, a message is printed on the console reporting the total number of PAM pages allocated to EAM. In this case, the message will read

EAM SPACE = 168 PAGES

However, if the request of 144 pages is only partially granted; say, only 100 pages are allocated, then the message will read, in this case

EAM SPACE = 124 PAGES

EAM space increases in this manner up to a maximum of 16384 PAM pages as long as system space is available.

On the other hand, EAM space decreases when EAM files are erased. Space is returned to the system in blocks of 24 pages when they are free. It is noted that EAM maintains a minimum space of 600 pages. In other words, EAM space does not decrease when total EAM space is less then the minimum. There is no message to the console for reduction of EAM space.

At an abnormal termination of a session, EAM saves all files required to be saved. In the meantime, the total EAM space allocated at that time is recorded in the system. Therefore, when the system is reloaded, whether it is a cold or a warm start, the same amount of space is given back to EAM and may be much higher than the designed minimum (600 pages). Due to the fact that the reduction of EAM space occurs when an EAM file is erased, the process of reducing EAM space will occur when the first EAM 'ERASE' is issued, such as at the end of printing or punching an EAM file. Therefore, for a warm start, EAM space will be reduced when saved spoolout from the last session is completed. However, for a cold start, EAM space will not be reduced until someone creates and then erases an EAM file.

Lastly, if EAM is unable to acquire more space (which implies that the public volumes are saturated) the operator is informed that EAM space is exhausted. The affected task will then be passed for 90 seconds if the operator so chooses. When control is regained by the task, the operation is repeated. Note that this procedure is continued indefinitely, and that control is not returned to the user program when space is exhausted.

FILE MAINTENANCE AND DISPOSITION

The facilities provided by VMOS for the maintenance and disposition of files reside in three functional areas. These areas encompass software-supplied program preparation and text editing routines, utility routines, and command language instructions.

Software-Supplied Routines

VMOS contains a variety of system resident program preparation and text editing routines that contain functions enabling the programmer to perform the following:

1. Alter the name of a source program or data file.

2. Copy a source program or data file from one file to another.

3. Erase all or part of program associated files (source, object, and data).

4. List the contents of a source program or data file.

The use of these routines is discussed in the VMOS Programmer's Reference Manual.

Utility Routines

The utility routines supported by VMOS enable the programmer to perform the following file maintenance functions:

1. Display and print PAM-formatted files, and manipulate the data and keys contained in such files.

2. Copy files from one medium to another: for example, card to tape, tape to tape, card to random access, random access to random access, random access to tape, and random access to printer or punch.

The descriptions of the routines that provide these facilities are contained in the VMOS Service Routines Manual.

Command Language Instructions

Through Command Language, VMOS supplies the programmer with the ability to perform a variety of file maintenance operations, and the instructional facilities for the disposition of files.

The sum of these operations encompass the following:

1. Catalog Entry Alteration,

2. File Definition Modification/Deletion, and

3. File Reproduction and Status Monitoring.

In addition, the programmer has the ability to manipulate certain system files.

Catalog Entry Alteration

VMOS supplies the programmer with two instructional sets for the maintenance of catalog entries: the CATALOG command (CATAL macro) and the ERASE command (macro). The CATALOG command and its associated macro enable the programmer to alter a file's catalog entry as follows:

1. Change the file's filename.
2. Specify a new catalog entry.
3. Associate read or write passwords with the file.
4. Specify the manner in which a file is to be processed.
5. Specify whether or not a file is sharable.
6. Specify a file's retention period.

A programmer may only alter files to which he has access.

The ERASE command permits the programmer to make the file logically empty, deallocate the space associated with the file, and remove the file's entry from the catalog. A programmer may only ERASE files created under his userid.

File Definition Modification/Deletion

The operating system supplies a set of four commands (macros) relative to the modification and deletion of a file definition: the CHANGE command (CHNGE macro), the RELEASE command (REL macro), DROP command, and HOLD command. These instructions affect only the definition of the file as related to program processing. The catalog entry for the file is not affected.

The CHANGE command and its related macro permits the programmer to change the linkname or symbolic device name associated with a previously defined file. This enables the programmer to pass FILE command definitions between programs within a task without changing the link symbols specified in the program's FCB's.

> Note: The CHANGE instructions operate through use of the file's linkname. Therefore, the instruction applies only to the last file definition preceding the CHANGE instruction.

The RELEASE command (REL macro), HOLD command, and DROP command enable the programmer to suspend the processing of, or delete, file definitions in a discretionary manner. The RELEASE command and its related macro permits the programmer to delete a file definition and, if he wishes, the devices associated with it. The HOLD command allows the programmer to temporarily remove a file definition from processing status. When used prior to a RELEASE directive, HOLD suspends the RELEASE. The DROP command removes a file definition from hold status. If a previously issued RELEASE directive for the file definition had been pended by a HOLD, the file definition would be released when a DROP is issued. Note that these three instructions operate, as did CHANGE, through the use of the linkname. They apply therefore to the file definition in effect at their issuance.

File Reporduction and Status Monitoring

The Command Language supplies the programmer with a COPY command and macro for the reproduction of files. These two directives copy files on a one to one basis. The COPY directives cannot be used to create a file with characteristics that are different than the file being copied. Also, these directives cannot be used to cause a file to overwrite itself. The use of the copy facilities are not restricted to files residing on like devices, that is, direct access files may be copied to tape as well as to other direct access devices. Tape to tape copying is not supported with this command (see Utility Routines discussion).

The FSTATUS command (FSTAT macro) enables the programmer to obtain a variety of information concerning the status of his file. The directive accepts either fully or partially qualified filenames. These directives permit the user to obtain information concerning the type of file, type of volume, space allocation, page utilization, security, file and volume characteristics, and password information.

System File and Task Library Reassignment

VMOS supplies the programmer with the ability to direct SYSDTA and SYSIPT files to different sources. These two input files normally have the same source as the SYSCMD file. However, using the SYSFILE command, the programmer can direct SYSDTA or SYSIPT to a cataloged file, a specific device, their primary source, or back to the SYSCMD file.

The SYSFILE command also enables the user to designate a file other than the system's TASK library as the TASK Library (TASKLIB) for the current task. The specified file must be an object module library created by the Library Maintenance Routine (see VMOS Service Routines Manual). This file will be used by the Linkage Editor and Dynamic Linking Loader to satisfy unresolved external references. If the programmer does not specify a TASKLIB file, the operating system assumes that none exists for the task.

FILE RECONSTRUCTION SYSTEM

Introduction

The VMOS File Reconstruction System supplies the programmer, in conjunction with the system controller, with two distinct functions for ensuring file integrity. The first function provides file reconstruction (via portion) following media damage. The second relates to software errors. If a file is updated with wrong information due to invalid data or erroneous processing by a user program, it may be necessary to undo the damage by resetting the file to its state at a time prior to the introduction of the error.

The function in the former case is referred to as file reconstruction; in the latter case as file resetting. File reconstruction requires a log of after images, that is, a copy of each record after it has been updated but just before it is written to the file. File resetting requires a log of before images, that is, a copy of each record just before it is modified or updated.

File Reconstruction (After Images)

If an attempt is made to read a record and the hardware detects an I/O error, it is necessary to reconstruct that record. A copy of that record must have been previously saved, or the file itself must be recreated from the original transaction data. The latter technique is appropriate and sufficient for most small files.

The File Reconstruction System (FRS) contains facilities using the former technique; that is, it maintains a copy, on tape, of all records output to the file; it creates an index to effect the rapid retrieval from tape of necessary record(s); and it places the copy of the record back into the file.

In order to maintain the copy, FRS (in conjunction with the access methods and Privileged PAM) will log to tape every PAM page written to the file. This service will be provided automatically to every file which has requested reconstruction copies through its Catalog Entry, that is, the system controller has issued an FRS command for the file, with the RECON=YES parameter.

If reconstruction services are requested after the file has been created, the entire file is logged to tape when it is next Opened. These after images are also generated automatically by FRS whenever needed.

A disc index is built and maintained by FRS to identify the log tape which contains each after image. When a record is modified (creating a new after image), references to any earlier after image of that page are ignored. A cleanup routine is supplied to the system controller to delete old after images from the log tape.

File Resetting (Before Images)

When a user program malfunctions, or invalid transactions are used to update a file, the file itself may become unusable. In order to reset the file to its state prior to the introduction of the error, FRS maintains a historical log of PAM pages in the file as they looked just prior to any modifications. These before images can be sequentially returned to the file, in effect, resetting the file backward in time.

It is not possible to reset just a portion of a file. For example, in ISAM each data page has specific relationship to the index pages, such that when one is reset, the other must be appropriately reset. In general, any file which contains embedded pointers must be reset consistently throughout the file.

Since ISAM and other access methods may delay the actual writing of index blocks beyond the time when the data pages themselves has been written, resetting a file requires that the reset operation continue until the index blocks do reflect the data pages. The only point at which this is guaranteed is at OPEN time. Thus, resetting a file to any given date/time requires that the file be reset to the first OPEN prior to that date/time.

Since the FRS is independent of specific access method characteristics, this requirement applies to all files, not just ISAM files. The FRS will locate the appropriate stopping point automatically, given a date/time (or OPEN count) by the user (see the RESET command).

If resetting services were requested after the file has been created, the file is assumed to have always previously been in that state. A reset request to an earlier date/time is effectively a reset request to the earliest OPEN time known to FRS.

When a before image log record becomes too old to be of any use to the file owner, as determined by him and the system controller, FRS will automatically discard it by deleting all reference to it in the disc index. Because of the requirement that the historical log be maintained through file OPEN time, all log records between an OPEN record (generated automatically) determined to be 'too old', and the subsequent CLOSE record are discarded. (The CLOSE record is also generated automatically by FRS.)

Log Image Tapes

The log file consists of a set of system tapes, referred to by unique VSN's, and a cataloged disc file which contains pointers to sets of log records on each tape.

The tapes on which all log records are placed are assigned to the FRS, not to the user. They are not known as files to the VMOS catalog management system, either individually or collectively. The log tapes are chained to each other in LIFO sequence, the most recently logged to tape pointing to its predecessor, and so on.

The FRS log tapes contain log records belonging to all users of FRS services. They may contain security-critical information, and certainly contain information which each user (and system controller) asserts is critical to the integrity of his data files. The FRS tapes should be handled carefully and safely stored.

User Interface

Most interfaces between the user, or user program, and FRS are between the user and Data Management System components, which validate and reformat the request for FRS processing.

No user is charged for services he does not require. The FRS command must be issued by the system controller before a file may use any FRS services, including the generation of log records on FRS log tapes.

Generating After Images

The generation of after images is automatic for any file for which the system controller has issued an FRS command with RECON=YES. No user or user program intervention is required, and no modifications need be made to existing user programs.

Whenever the access method attempts to write a PAM page to a file requiring after images, the write operation is 'intercepted' by FRS and the log taken.

If an already existing file is authorized for after image logging, the entire file will be logged to tape at the time it is opened, before control is returned to the user program.

After images will also be generated automatically for any CATALOG command/macro, FILE command/macro, and ERASE command/macro issued for the file.

Generating Before Images

Although all after images necessary for reconstruction will be generated automatically (as far as the user program is concerned), before images necessary for file resetting may require the cooperation of the user program. This is a function of the access method used to process the file, as well as the mode in which it is opened.

Move Mode (SAM,ISAM)

If a file is opened in move mode, the access method assumes full control over the file's I/O areas, or buffers. (To be precise, the access method has control over the placement of logical records into the buffers. The buffers themselves may be independently accessible to the user program.) Only SAM and ISAM permit this mode of OPEN.

Since the access method 'intercepts' each output-type operation, it can easily detect that a PAM page (or block of pages) is about to be modified. At this time, before the modification is permitted, the access method requests a before image from FRS. The page can then be modified.

Before images for SAM or ISAM files operating in move mode are produced automatically without requiring user program concern.

Before images will also be generated automatically at the time the file is opened and closed. These special before images are used during reset operation to control the starting and stopping points of the reset.

Locate Mode (SAM, ISAM)

If a file is opened in a locate mode, the access method provides only the location, within a buffer or I/O area, where the user program may modify the logical record. Both SAM and ISAM permit this mode of OPEN. The access method, and FRS, are unaware of any independent action the user program may already have undertaken to modify the buffer.

The user program must therefore undertake to provide the indication of intention to modify, by use of the LOG macro.

Locate Mode (User PAM)

User PAM functions only in locate mode. The user program informs the access method (and FRS) that it intends to modify an already retrieved PAM half-page, using a PAM action macro with the LOG operand.

Considerations for Generation of Before Images

Although the generation of before images is automatic for move mode, and essentially straightforward for locate mode, the user should be aware of actions taken for him by the access method under certain conditions.

In the case of SAM and ISAM file, before images will be taken of the entire buffer, not just the individual PAM page being modified. However, the access method will inhibit multiple logs of the same PAM page, even though the user may read and then modify the page more than once while the file is OPEN.

An additional anomaly occurs in the case of SAM files. When a SETL action macro is issued to reposition the file, and the file is opened OUTPUT or EXTEND, all PAM pages between the current end of the file and the new position are logged as before images. This action is automatic.

Guaranteeing Before Images for Sharable Files

File reconstruction and resetting services can be provided automatically, without program changes, for any file opened in move mode. If locate mode is specified, user macros will have to be added to guarantee accurate reset logs (that is, before images).

What is critical is that the owner of a file has little control over the manner in which authorized users access his file. The owner cannot therefore guarantee that, no matter who uses the file, his before image logs are correct.

The MODE parameter of the FRS command (see VMOS Installation Management Manual) will provide greater assurance of accurate resetting.

$$\text{MODE} = \left\{ \begin{array}{l} \text{MOVE} \\ \text{LOCATE} \end{array} \right\}$$

If MOVE is specified, the file can only be opened in move mode by any authorized user. If LOCATE is specified (or defaulted), the file can be opened in either move or locate mode.

The user is then able to opt for an accurate backup, at the expense of preventing locate mode access to the file (with SAM and ISAM), and by preventing the use of user PAM completely.

Inserting User Information in Log Images

Each before or after image log record contains a 16-byte field for user-specified information (see below). This field will not be used by FRS at any time, although installation-dependent routines (which might require this information) may be written to read and use the log tapes.

User Options for Unrecoverable I/O Errors

When an unrecoverable I/O error occurs, the access method and privileged PAM will place into privileged extension of the FCB a range of logical PAM pages which need to be reconstructed.

If FRS services have been requested for the file, the access method will give control to the user at his RECON exit. This is in addition to other EXLST macro parameters. (See EXLST macro.)

At this time, the user can issue a RECON macro, or continue processing.

If the RECON exit of the EXLST macro is NO or defaulted, control is passed to the ERRADDR exit. The RECON exit is not used if FRS services are not requested for the file.

Restrictions Resulting from Use of FRS

Once the system controller has authorized reconstruction or resetting services to be made available for a file, the user is not allowed to ERASE the file or to open the file OUTPUT or OUTIN. It is reasoned that a user does not wish to destroy too easily that which he has carefully prepared.

Therefore, in order to ERASE a file for which FRS services are used, or to reopen it OUTPUT or OUTIN, the Catalog Entry must first be changed by the system controller with:

```
/FRS        filename,RECON=NO,RESET=NO
```

It is impossible to recover the file with FRS if it is destroyed in this manner.

Note that the first OPEN OUTPUT or OUTIN is allowed, so that the file may be created.

FRS Tape Record (LOGTAPE) Formats

Figure 2-10 illustrates the format of the FRS Log Tapes.

The standard volume label (SVL) is the initial record on each log tape. The volume label number is always one.

The standard file header label number is also always one. The file-identifier always contains the name SYSLOG.TAPE. The volume sequence number, file sequence number, generation number, and version number are also always one.

The volume serial number in the SVL and the file serial number in the HDR1 label are always the same and should be the same as the external surface number of the tape for visual identification.

The expiration date is always the same as the creation date.

The block count field in the EOV label contains the number of data blocks between the header label group and the trailer label, excluding tape marks (TM). The field is in zoned-decimal form.

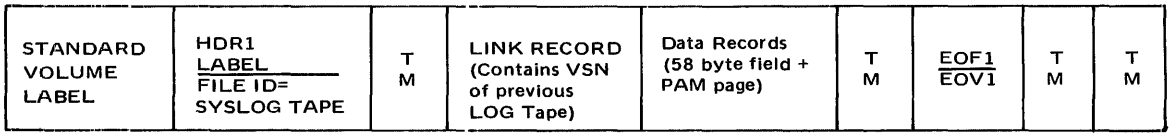


FIGURE 2-10. FORMAT OF FRS LOG TAPE

Linkage Record (80 bytes)

The Initial data record on the log tape is a linkage record used only by the FRS. It is, however, counted in the tape's block count. The format of the linkage record is shown below.

Field	Bytes	Length	Description
1	1-4	4	Label identifier. Contains LNK1 to indicate that this is a linkage record.
2	5-10	6	Predecessor volume serial number. This field contains the volume serial number of the log tape which preceded this log tape. When the current log tape is the initial one in the series, this field is set to binary zeros.
3	11-80	70	Not used. This field is set to blanks.

Before and After Image Records (2106 bytes)

Field	Bytes	Length	Description
1	1-13	13	Date-time stamp. This field is in the form as follows: YYMMDDHHMMSSS, i.e., date, followed by time of day.
2	14	1	Log record control byte. This field indicates the type of log record. A byte of X'1C' indicates a Before Image; a byte of X'20', an After Image.
3	15-18	4	Task sequence number. This field contains the TSN of the task which requested that the log record be created.

Field	Bytes	Length	Description
4	19-26	8	User ID. Userid of the requestor is contained in this field.
5	27-42	16	User information. This field contains any information which the requestor wishes to add to the log record. If nothing is added, this field is set to blanks.
6	43-58	16	PAM key. This field contains the key portion of the half-page being logged.
7	59-2106	2048	PAM half-page. This field contains the PAM half-page being logged.

Catalog Entry Image Records (2106 bytes)

These records are the same in format as the Before and After Image records with the following exceptions:

Field	Bytes	Length	Description
2	14	1	Control Byte. Refer to Log Record Control Byte.
5	27-42	16	Not used. This field is always set to blanks.
6	43-58	16	Coded File Id. Only the first four (4) bytes of this field are meaningful. They contain the coded file Id assigned to the file by the DMS. The last 12 bytes are binary zeros.
7	59-2106	2048	Catalog Entry. This field contains the catalog entry of the file, left-justified with trailing X'00's.

The last log record is followed by a tape mark.

The EOF Standard Tape File Label follows this tape mark. The EOF label is followed by two tape marks.

Description of FRS Printout

There are four types of printouts to inform the user of the action taken by the recon-reset or cleanup processor.

TYPE 1 – Parameter File of Operations

At the beginning of the recon-reset or cleanup process and at the end of each log-tape, all operations are listed with their status.

TYPE 2 – Records of Action

Each time an output operation is initiated, a concise record of the action taken is listed.

TYPE 3 – Record totals by LOG-TAPE

At the end of each LOG-TAPE, totals are listed as to the number and type of records used both input and output.

TYPE 4 – Error Messages

When an error is encountered, a message is listed to notify the user of the specific error type and the operation is marked so that the status will reflect an error condition.

TYPE 1 – Parameter File of Operations

This printout may be subdivided into three general categories:

1. Heading lines
2. First line of operations
3. Succeeding lines of operations

HEADING LINES:

The parameter file list begins at the top of a page with the first line identifying the date, time, type of process being performed (e.g., reset-recon or cleanup), and the point within the job being performed (e.g., before processing, end of VSN x, etc.).

The second line of header identifies the data fields in the first data line (e.g., Password, User ID, etc.).

FIRST LINE OF OPERATION

The first data line consists of five fields

STATM.# – This field is a sequential number assigned by the program to each operation. This statement number is referenced by Error Messages and Records of actions.

Type of operation being performed. There are five operations currently supported.

RECON-Reset Processing:

RESET= Reset a file to a specific open-close time

RECON= Reconstruct a specific frame

RECONALL= Reconstruct a complete file

Clean-Up Processing:

CLUP-AFT= Cleanup after image by deleting all after images from log-tapes. Then take an initial copy of all files requesting reconstruction services.

CLUP-BEF= Cleanup before images, saving all records between appropriate OPEN-CLOSE pairs in the index.

The file PASSWORD is printed in hexadecimal format.

User Identification and filename.

The CODED-FILE-ID that was assigned to the file when initially cataloged is printed in hexadecimal format.

Status of Operation:

NOT COMPLETE = The operation is not complete

COMPLETE= The operation is complete

The following may appear with either of the above, under special conditions:

ERR= An error was encountered when processing this operation. The specific type of error that occurred is documented by a TYPE 4 error message.

DUPL= The parameter file is specifying, for a given log-tape, a duplicate operation to be performed for a specific frame.

OPEN= This specifies a RESET or CLEAN-UP-BEFORE operation has begun on this file but the OPEN or CLOSE log-record (depending on the operation) has not been encountered to signify its error free completion. (Example – if a read error was encountered on an OPEN record (and bypassed) which was needed to complete the reset operation, the OPEN flag would be printed.)

SUCCEEDING LINES OF OPERATIONS:

The contents of the second and succeeding lines depend on the type of operation to be performed.

RESET — The second line of a reset operation is as follows:

CLOSE DATE mm/dd TIME hh/mm OPEN DATE mm/dd TIME hh/mm

RECON — The third and succeeding lines are lists of all VSN's which will be processed for this operation. The second and succeeding lines define the VSN's to be processed and the specific frames within the VSN. Preceding each frame number and separated from it by a comma, the status of the frame is printed.

N = NOT COMPLETE
L = COMPLETE
D = DUPLICATE
E = ERROR

RECON-ALL — The second line of a reconstruct all frames is as follows:

LOW-FRAME = n HIGH-FRAME = n
FRAMES NOT COMPLETE = n

where n = 1 to 7 digit count

The third and succeeding lines specify the VSN's to be processed for this operation.

CLEAN-UP BEFORE — The format is the same as the RESET format.

CLEAN-UP AFTER — Each line of this operation identifies files that are to be initially copied. It specifies its User ID and filename, password and status.

TYPE 2 – Records of Action

Before each output operation is performed, a record of the operation is listed. This record contains the number of the statement (STM,#=n) which called for the action, followed by the frame to be written (FRAME=n). The frame number is a 6-digit hexadecimal field specifying the half-page to be updated. If the action is to update the file catalog entry, the word 'OPEN' or 'CLOSE' is substituted for the frame number. To minimize the number of pages of printout, up to five records of action are listed on one printer line.

Note: If an output operation is not successful, an error message will follow the Record of Action specifying the type of error.

TYPE 3 – RECORD Totals by LOG-TAPE

At the end of a LOG-TAPE, totals are listed as to the number and type of records, both read and write, along with the number of input and output errors.

The types of totals accumulated are:

1. Total number of records read from the LOG-TAPE
2. /CAT = Catalog commands/macros
3. /FILE = File commands/macros
4. /ERASE = Erase commands/macros
5. OPEN-A = Open after images
6. OPEN-B = Open before images
7. CLOSE-A = Close after images
8. CLOSE-B = Close before images
9. OPEN-FRS = Reserved
10. CLOSE-FRS = Reserved
11. LOG-B = Log records of before images
12. LOG-A = Log records of after images
13. ERRORS = The number of errors encountered

All of the above totals are accumulated and printed for input. Only types 6, 8, 11, 12, and 13 are accumulated for output.

> Note: If all operations are complete for a given VSN before the end of tape is reached, the end of VSN is forced and a special heading is printed before the totals.

TYPE 4 – Error Messages

LOG-TAPE INPUT ERRORS

When an input error is encountered, the operator is informed of the DMS error code. If he returns control to the program and the error is a read error, the record is bypassed and printed. All other errors cause a message to be typed to the operator, if batch, and to the terminal, if interactive. It specifies the type of error (DMS error) and gives the option to Retry (R) or Terminate (T).

OUTPUT ERRORS:

There are three basic phrases printed when an error condition is encountered.

"DMS ERROR v v WHEN PROC.STM# z"

where:

x = DMS error code (IDEMS)

y = 'blank' normal writing mode (only that record lost)

= 'ALLOCAT' while allocating (Reset operation not processed)

= 'FSTATUS' while performing FSTATUS (Reset operation not processed)

= 'OPEN' while opening a file for clean-up after (operation not processed)

= 'CLOSE' closing a file for clean-up after (operation may be OK)

= '$RDCT')

= '$FRDT'> *

= '$WTCT')

= '$AQIR')

z = Statement number

"FILE-NAME = Filename"

The filename is appended to the first message when processing clean up after.

"RECORD WAS NOT OUTPUT SUCCESSFULLY"

This message is appended to the first message when a data record (frame) cannot be output.

*When updating a catalog entry while processing a reset – (catalog entry may contain invalid last page number or last record pointer).

INTRODUCTION

The VMOS Command Language supplies a programmer with the ability to acquire private devices for his task, to assign memory areas for his programming to regulate the use of devices, and to deallocate the devices and memory he has acquired.

The system, through the operation of its control program or in conjunction with the operator, performs certain functions and imposes conditions with which the programmer should be familiar. These have been outlined for the reader prior to the discussions of device and space allocation, regulation, and disposition.

All VMOS directives referred to in this section are described in detail in Section 3 of Part 3. Refer to the appendices for parameters and I/O features (under "Macros Supporting Physical Level I/O and Run Time Parameters") and/or TOS Commands and Macros (under "Miscellaneous Task and File (Data) Management Commands and Macros ").

System Device Allocation Component

The function of the system's device allocation (DA) component is to provide tasks with the devices required for I/O processing. The allocation of peripheral devices is under direct control of this VMOS component rather than under system operator control. The console typewriter is not controlled by device allocation. The console is a system resource and is considered to be available to any task.

Device allocation maintains a list of the devices which are currently available to the system. The maintenance responsibility for this list, in certain circumstances, extends beyond the DA component, as in the case of multireel tape files. DA processing routines are nonresident system routines which are function attached by their calling tasks. The various control tables required by the DA component reside in system memory.

System Device Accounting

The system maintains the number of requests for input/output service in the task's TCB (task control block). This is a running total and reflects, at any time, the number of I/O requests made by the task. This value includes I/O service for purposes of error recovery.

The system also accounts for the total number of minutes a device was assigned to a task. Only private devices are included in this accounting. The system maintains this information in the task's TDT (task device table).

System Device Control Optimization

Through operation of its device control activities, the system ensures that all direct access seek operations are performed off-line. If the seek operation was the initial command in the chain, the system automatically initiates the remainder of the chain after the seek has been completed. If the seek was the only command in the chain, no further action is taken by the system when the seek completes.

If a user's random access chain does not begin with a seek, the system executes the commands in his chain in-line; that is, if any seek commands are in the chain, they do not proceed off-line, but rather, keep the control unit busy until the seek operation completes. Note that all Control Program and PAM random access command chains begin with a seek; the exception would be Class I problem programs utilizing TOS volumes.

In addition to ensuring that seek operations are performed off-line, the system manages the movement of seeking mechanisms (arms) for public volumes to minimize their seek times. The arms traverse the volume from lowest to highest cylinder and then return. The volume is viewed as a series of contiguous sectors, each sector being composed of eight cylinders. A request falls into one of 26 sections and is processed on a FIFO basis within its sector. This strategy ensures that, once in a particular sector, the worst case arm movement will involve 7 positions. A sector is serviced for n requests. Any remaining requests are processed on the return pass. This prevents the sectors at either extremity of the disc from being penalized.

The processing of off-line-seeks takes precedence over the transfer of data as long as there is a unit which requires off-line-seek service. Once all arms, which should be, are seeking, the transfer of data will begin as soon as the first seek-complete interrupt is recevied.

Operator Interaction

When a file residing on private volumes requires a device, the system sends a mount message to the operator. This message specifies the VSN of the volume to be mounted and suggests a specific device (and bin number for an MSU) on which to mount the volume. The operator can honor the request, decline the request, or specify a different device (and bin number for an MSU) on which to mount the volume. If the operator specifies a device (and bin number for an MSU) which is available and which is of the same device type, the system accepts the recommendation. Otherwise, the operator's request is rejected and he is again prompted to mount the volume.

The system does not necessarily issue mounting instructions in every case. Since VSN's of the volumes currently on line are known by the system, acquisition of these volumes can occur without operator assistance. Note that when a tape device is deallocated (for example, at LOGOFF, via the RELEASE command) the VSN is removed from the system's VSN table.

For TOS volumes, the operator is asked to mount SYMDEV= (symbolic device) on a device (and bin for an MSU). Again, the operator can honor the request or decline the request. If he specifies a device (and bin number for an MSU) which is not available, another mounting message is issued.

Note: When a mass storage volume is being directed to a particular MSU device, the operator replies in the form NN(b) where NN is the name of the device and b is the bin number. The bin number can range from 0 to 7.

Multichannel Switch

When a VMOS system is generated, random access devices which are to be shared by two processors are marked as such in the Physical Device Table. Whenever I/O is to be performed for these devices, Device Control attempts to reserve the device for its processor. If the multichannel switch is being used by the other processor, Device Control notes this fact and prepares for the reception of an interrupt from the switch when the other processor's operation completes. Once the switch is available to the requesting processor, the attempt to reserve the device is made. If the device is reserved by the other processor, error recovery is invoked, and a retry is made at a later time. Refer to "Device Management Control Tables" in the Appendices for further information.

Once the switch and device are both available, the off-line seek operation is initiated. When the seek operation is completed, Device Control initiates the data transfer operation. This command chain is preceded by a 'release' command, which, at the completion of the channel operation, leaves the device free to be utilized by the other processor.

Note that the processor is expected to release the device when it is no longer needed. This is always done by VMOS, as noted above. If the processor is operating under a system which fails to do this, the other processor is unable to access the device.

The reservation and releasing of devices takes place on each input/output operation. This level of support, therefore, does not permit one processor to retain control of a particular device through a series of I/O operations. Once an I/O operation is completed, the controlled device is available for reservation by the processor which asks for it first. Some DMS functions, such as Cataloging, and Allocating space, require the execution of multiple I/O operations to complete their function. During these processes, the data being manipulated must not be accessed by another processor, or the state of that data will become indeterminate. The level of support for the MCS does not guarantee the lockout of a device across a series of I/O operations.

DEVICE ACQUISITION

Public devices, those identified as being accessable concurrently to all users, cannot be restricted by the programmer to the use of his task alone. A programmer may require only private devices, that is, devices for files not on public volumes. The acquisition process constitutes identifying and obtaining the device and associating it with a file.

Device Identification

In order to acquire a peripheral device to be used by his task, a programmer must identify the device to the system. Functionally, device identification operates on three levels: unit, family, and type. The programmer may, then, acquire a specific device unit by specifying the installation mnemonic for the device, or he may acquire a device of a specific family or type. The risk factor associated with successful device acquisition varies directly with the degree of specification. Acquisition by device mnemonic carries the greatest risk of failure; acquisition by device type, the least.

The association of a device with a file (FILE command) can only be done at the family and type level. For example, a programmer can associate a file with a non-return-to-zero 9-level tape or with any 9-level tape. However, the acquisition of devices for a task (SECURE command) may be accomplished at all three levels of identification. It is the programmer's responsibility to assure that proper device-file association is maintained, for unles otherwise directed the system assigns devices to files sequentially. In other words, the first file identified to the system gets the first device secured unless instructions to the contrary have been issued.

Obtaining Devices

Devices are obtained through use of the SECURE command. This command enables the programmer, as a prerequisite of task scheduling, to reserve the resources that a task will require for its execution. These resources remain in possession of the task until it is terminated, or until the programmer explicitly releases them. Using the SECURE command, a programmer may reserve either specific device units or devices belonging to a given family. If an additional SECURE is issued while a previous SECURE is still in effect, the task will be terminated by the system.

Unsuccessful attempts to secure devices are resolved in one of two ways depending on whether the task is conversational or background. An unsuccessful attempt to obtain devices for a conversational task will cause a message to that effect to be sent to the user's terminal.

The programmer may then issue another SECURE command for a different device or allow the operator to assign whatever device of the required type is available when the device is called for during program execution.

If an unsuccessful SECURE occurs in a background task, a message to that effect is sent to the task's diagnostic file, the command is ignored, and the operator will assign a device of the type required during program execution. It is therefore advisable not to attempt to secure specific device units for background tasks, since an unsuccessful attempt could result in an unacceptable device being assigned to a task.

The SECURE command may be used for tasks processing either Class I or Class II programs. A programmer may also request specific devices for TOS files by using the TOS FILE command. Similar to the processing of unsuccessful SECURE commands, the system will alert the operator to assign another device, of proper type, if the one specified by the TOS FILE command is unavailable.

Remote Batch Work Stations

Remote batch work stations are logically attached to the Remote Batch Processing (RBP) system by the RSTART command. This command identifies the work station and defines its hardware configuration.

Device Assignment

Device assignment in VMOS is the term applied to the association of a private device with a file. For all Class II programs and most Class I programs, device assignment is accomplished by use of the FILE command. This command enables the programmer to assign devices of the family or type level for his files. If no device assignment is indicated to the system, the system associates the file with public volume devices. If no FILE command is present, the logic used by the system's OPEN processing will request device assignment.

The programmer is cautioned to secure all devices required for his task prior to attempting to perform device assignment, because the system will suspend the processing of a task when no more devices are available during processing. In other words, the system will attempt to assign devices as they are requested, but will suspend the requesting task when no more devices are available. The suspension will remain in effect until the requested device becomes available. This situation could result in poor turnaround time for tasks requiring several devices.

Class I tasks may also request assignment during task execution, via TOS ASSGN macro instruction for TOS files. This same macro is also used by TOS logical level I/O dynamic assignment. For files processed at the physical level, device acquisition is performed when the first EXCP macro instruction is executed.

SPACE ACQUISITION

When a programmer initiates the processing of a program, the operating system automatically allocates the task enough memory to begin processing. Additional space will be allocated to the program as required within the limits imposed by the system.

A programmer may impose certain conditions on the allocation of space for the processing of his program within the limits imposed by the system. Using the REQM macro, the programmer can request, in multiples of one page, that contiguous memory be obtained for his program. REQM cannot be used to preallocate space; it is issued during program execution, and, unless the program contains an instruction (RELM macro) explicitly releasing the space, it is returned to the system at program termination. Class I programs are allocated physical memory; Class II programs obtain virtual memory.

The programmer may also, when calling for the execution of his program, alert the system that the program will require a certain amount of space. This is accomplished using the CLASS parameter of the EXECUTE command, and enables the programmer to indicate the minimum and maximum memory requirements for a program. This instruction does not however acquire space; it only indicates the memory requirements, within the limits of the space available to the user, that a program may require.

The maximum public file space (direct access storage space) available to a user is specified when the user is joined to the system. This public space limit is not extended dynamically by the system. If a program attempts to create a file in public space larger than the space available to the user, the system will notify the program with an appropriate error code, and terminate the task if the program contains no contingency routine to deal with the situation.

Prior to program execution, the programmer can make more space available for his files by requesting extra space from the system controller, or by erasing existing files no longer needed (ERASE command). File space can also be freed during program processing, either from the command stream (ERASE command) or as a part of the program's execution logic (ERASE macro).

Within the limits of the space available to him, the programmer may allocate direct access space for a file using the FILE command or macro. The SPACE parameter of this instruction enables the user to declare the number of half-pages to be initially assigned to his file, the number of additional half-pages to be assigned when the initial allocation has been exhausted, the total number of half-pages required without regard to initial or secondary allocation, and to specify that a volume be allocated completely to file (ABS parameter). Absolute allocation on public volumes can only be one by the System Controller; the programmer is restricted to private volumes for absolute allocation. The programmer may, of course, not call for space in excess of the remaining space available to him. If this is attempted, the system will terminate the directive, and so notify the user. If the directive is entered in a background task, the request will be defaulted to the system standard.

DEVICE AND SPACE REGULATION

Except for the manipulation of the system input files, device and space regulation is a function of how the device and space acquisition and file management instructions described earlier are employed. For instance, the assigning of devices secured for a task (SECURE command) to specific files (FILE command) may be considered device regulation. For, if the device-to-file association were not specified, the devices would be assigned to the various files according to the order in which the devices were secured and the sequence in which the files were accessed. In other words, the first file for which a FILE command was issued would be assigned the first device secured. Similarily, space regulation may be viewed as the operation performed by the REQM macro because the programmer is designating both the amount of space he wants acquired and the way in which he wants it structured. The same analogy can be drawn for the use of the CLASS parameter of the EXECUTE command.

System Input Files

In addition to directing SYSDTA or SYSIPT to a cataloged file or the command stream, the SYSFILE command can be used to assign these input files to a specific device. The device can be any system supported input device as identified by its installation mnemonic. The card reader may also be indicated by the CARD option of the command.

DEVICE AND SPACE DEALLOCATION

As devices are assigned to a file only for the duration of the task in which assignment is requested, device deallocation always occurs as a result of task termination. However, the programmer can explicitly disassociate private devices from a task during program processing. This occurs when the programmer deletes a file definition using the RELEASE command or REL macro. These instructions permit the programmer to either release the device for further use by his task even though the file definition is deleted.

The user's Class II problem program may release devices when a file is closed via the DISCON parameter of the CLOSE macro. Devices assigned to Class I tasks may be released by the program's issuance of the TOS DDEV macro instruction, TOS Close logic, or VMOS Close logic.

Remote batch work stations are logically detached from the RBP system by the RSTOP command. Prior to detachment, all queued messages to the work station are transmitted, and the last message transmitted indicates that the work station has been detached. Subsequently, no output will be returned to the work station until it is once more attached to the RBP system.

File space on direct access devices may be deallocated using the SPACE parameter of the FILE command. To accomplish the reduction of space allocated to a file to the actual space occupied by the file, the programmer can issue a FILE command against the file and specify a negative primary space allocation equal to the difference between the actual file size and the original primary allocation figure, and a secondary allocation of zero. The actual space occupied by the file can be ascertained using the FSTATUS command. The programmer should bear in mind that the system always rounds space requests (positive or negative) to the nearest multiple of 3 pages. Therefore, it is possible to unwittingly erase data when calling for the deallocation of file space using the SPACE parameter of the FILE command. For example, if the programmer specified an initial primary space allocation of seven pages, nine pages would be allocated by the system. Conversely, if the file actually occupied only four of the nine pages, a FILE command to delete five pages SPACE=(–5,0) would result not in a file of four pages but rather in a file of three pages. Hence, a loss of data.

Processing space (virtual or physical memory) is returned to the system when task terminates. The user may, however, release space during processing by using the RELM macro. This macro enables the user to release contiguous areas of real or virtual memory, depending on the type of program (CLASS I or II) in operation. Although, this macro is functionally the obverse of the REQM macro, it may be used to release memory allocated to a program that was not obtained by user-initiated REQM. For instance, a RELM macro may be issued to release memory obtained by a program in excess of that which was required for initialization. Finally, a RELM may be used to release memory obtained by more than one REQM provided the separate memory areas are contiguous.

GENERAL

VMOS supplies an interactive user the ability to communicate with one or more terminals through use of the Communications Access Method (CAM). CAM is a set of routines residing within the Control Program and is designed to facilitate the implementation of inquiry/response programs. Programs making use of the facilities of CAM are written using the macros described in this section. These programs reside within the VMOS task environment and may be executed conversationally or nonconversationally. CAM programs, therefore, have access to all VMOS facilities including the use of system files. Furthermore, the subroutines which constitute the CAM facility are reentrant, and thus support concurrent execution of CAM programs.

The terminals supported by CAM are the 8752 and 7522 Video Data Terminals, the AT&T 33, 35, and 37 Teletypes, and the IBM 2741 Communications Terminal. The 8750 Video Data System is also supported for polled terminal applications.

PROGRAM AND TERMINAL IDENTIFICATION

The facilities for identifying terminals and programs to the CAM subsystem are supplied by the CONNECT command and PRIME macro. The CONNECT command enables a terminal-based user to converse with a program utilizing CAM. CONNECT performs the same function in respect to CAM programs as does LOGON in respect to normal interactive operation. The CONNECT command also provides the user with the ability to detach his program from CAM and return control to VMOS. In addition, CONNECT enables the user to transfer from one CAM program to another.

The PRIME macro is the directive used to specify the name by which the user identifies his program to CAM. This is the first indication to the system that a program is to run under CAM. The name which the user specifies with PRIME is the name that must be used by the terminal-based user when he enters the CONNECT command. A CAM program may be identified by only one name, that is, only one PRIME macro may be issued from a program. This macro also enables the programmer to define the size and number of entries to be created in the program's buffer pool.

MESSAGE PROCESSING

The processing of messages to and from CAM is accomplished using two sets of macros. The FETCH and SEND macros supply the means for message transmission. The FORM, NOLNS, and LNTYP macros define the message structure.

Messages are received by CAM from terminals via the FETCH macro. FETCH first ascertains whether an input message exists. If a message has been sent to the program, FETCH places it into a buffer specified by the program. If no message exists, control is returned to the CAM program. SEND is the macro employed to transmit messages to terminals. SEND operates in conjunction with the Event Control Block which, in addition to other functions, enables the user to select transmission options. These options permit the user to have a read issued to a line after a write is completed and/or free the buffer containing the message.

The FORM macro is used to format a message to be transmitted by SEND into the buffer acquired from the buffer pool. The NOLNS macro counts the number of lines (remote terminals) connected to a CAM user program. Since a connected line may be active or inactive, the number of lines recognized by NOLNS reflects both the number of active lines and the total number of lines connected to CAM. The LNTYP macro supplies the user with the device type code of any number of specific lines or a complete list of line numbers and their associated device type codes, and a count of the number of connected lines as supplied by NOLNS.

TRANSMISSION AND BUFFER CONTROL

Transmission and buffer control are supplied by a set of six macros. FBUF and GBUF are used to release and obtain buffers. WAITM, CAMRD, WREND, and SHUTM control message transmission.

The GBUF macro is used to provide buffer space from the buffer pool. FBUF provides the means to return buffers to the pool. When FBUF is used, the buffers must be released in the same order as they were obtained by GBUF.

The WAITM macro is used to pend a task until input is supplied from one of the terminals. WAITM should be used when the program is unable to receive further input. CAMRD is used to put up a read to a specified terminal. CAMRD does not accept data from the terminal, it only notifies the terminal that CAM expects the terminal to send a message. The SHUTM macro, like WAITM, is used to inhibit further input. However, SHUTM does permit the CAM program to send messages to the terminal. SHUTM should be used when the program is preparing to terminate. The WREND macro provides the user with the ability to stop processing until the SEND he initiated has completed. After the SEND is complete, control is returned to the user.

MULTISTATION LINE USAGE

The SPSEQ, ACT, and DEACT macros supply the user with the ability to use multistation lines with his program. SPSEQ is used to specify to CAM the multistation lines to be connected to the program and to supply CAM with the desired polling sequence for those lines. ACT and DEACT enable the user to activate or deactivate specific lines connected to this program. Specifically, DEACT is used to exclude a line from having data transmitted over it or to prevent any terminal from sending or receiving data. ACT is provided to restore inactivated lines (terminals) to the state of being able to receive or transmit data.

Part 3
Control System Commands and Macros

GENERAL

The four sections constituting this Part provide detailed descriptions of the commands and macros available for task, file, and device management along with the instructions necessary to implement the use of the VMOS communications facilities. The task, file, and device management sections have been individually prefaced with summaries relating groups of commands/macros to more selective functional areas such as task initiation, file creation, and device allocation. These summaries correspond to categories presented in the decision tables of Part 2.

This Part is intended to be used as a reference work in conjunction with the generalized discussions of task, file, and device management contained in Part 2. Each command/macro is treated as a separate entity, and, except for those instruction sets which are wholly independent, the description of each command/macro relates only to the function it performs.

GENERAL

The information contained in this section defines the explicit instructions provided by VMOS for programmer control of the task and program environment. The following summaries group the commands and macros into functional sets: task initiation, program initiation, task regulation, program direction, task termination, program termination, and process interruption.

Commands relative to remote job (Remote Batch System) initiation and termination are included in the corresponding task command summaries. Where commands or macros apply to both the task and program environments, they will be listed in all related summaries.

TASK INITIATION COMMANDS AND MACROS SUMMARY

Command/Macro	Function
DO	Execute a procedure file.
ENTER Command/Macro	Initiate a nonconversational task conversationally.
LOGON	Initiate primary task.
RLOGON	Identify user to RBP system.
RJOB	Identify and define processing for jobs submitted to the RBP system.

PROGRAM INITIATION COMMANDS AND MACROS SUMMARY

Command/Macro	Function
DO	Execute a procedure file.
ENTER Command/Macro	Initiate a nonconversational task conversationally.
EXECUTE Command/Macro	Load a program into memory and call for its execution.
LOAD Command/Macro	Load a program into memory.
PROCEDURE Statement	The first entry in a PROC file. Indicates printing options.

TASK REGULATION COMMANDS AND MACROS SUMMARY

Command/Macro	Function
RSTATUS Command	Determine status of one or more remotely submitted jobs.
SETSW Command	Set, reset, and invert task switches.
SKIP Command	Test task switches.
STATUS Command	Determine the progress through the system of tasks being processed.
STEP Command	Define logical subdivisions of a task.
TMODE Macro	Provide task information.

PROGRAM DIRECTION COMMANDS AND MACROS SUMMARY

Command/Macro	Function
CALL Macro	Branch from main program to subroutine.
CHKPT Macro	Permit user to specify node points at which system environment and program status are to be noted.
CSTAT Command	Change status of pages owned by a Class II program.
EOF Command	Pass control to programs end-of-file address. Transfer out of a conversational task loop.
EXCOM Macro	Get data from Common Data Area.
EXECM Macro	Terminate a program and initiate execution of another.
GETSW Macro	Store settings of the 32 program task switches.
LOADM Macro	Terminate a program and load a new program.
LPOV Macro	Cause a load module to be loaded during program execution. See Linkage Editor Description in VMOS Service Routines Manual.

Command/Macro	Function
PARAMETER Command	Indicate options available to language processors. See VMOS Program Management Manual.
PASS Macro	Relinquish remainder of time slice.
PROUT Macro	Cause a record to be written to the SYSLST file. (TOS Support.)
RDATA Macro	Retrieve next record from SYSDTA.
RDCRD Macro	Get an input record from SYSIPT. (TOS support.)
REMARK Command	Allows user to output remarks to SYSOUT file.
RESTART Command	Reload a program interrupted by a checkpoint at the point at which the checkpoint was taken.
RETRN Macro	Return control to a calling program from a called program.
RMSG Command	Send a message to another RBP user, the central operator, or an RBP work station.
SAVE Macro	Store the contents of general registers.
SETSW Macro	Set, reset, and invert task switches.
TOCOM Macro	Move data to Common Data Area.
VPASS Macro	Relinquish control of the processor.
WRLST Macro	Cause a record to be written to SYSLST.
WROUT Macro	Send a message from a program to SYSOUT.
WRTOT Macro	Write a record to SYSOPT (TOS support).
WRTRD Macro	Send a message to a terminal and accept a response from a terminal.

TASK TERMINATION COMMANDS AND MACROS SUMMARY

Command/Macro	Function
ABEND	Force abnormal task termination, obtain a PDUMP of class 5 and 6 memory, and be automatically logged-off.
CANCEL	Cease processing a secondary task. Transfer out of a conversational task loop.
ENDP	Return control from a procedure file to the primary command input stream.
LOGOFF Command/Macro	Terminate primary task.
RLOGOFF	Indicate completion of an RBP session.
ROUT	Retrieve deferred or discontinued remote job output (RBP).

PROGRAM TERMINATION COMMANDS AND MACROS SUMMARY

Command/Macro	Function
ENDP Command	Return control from a procedure file to the primary command input stream.
TERM Macro	Terminate a program.
TERMD Macro	Terminate a program and obtain a program dump.
TERMJ Macro	Terminate a job within a task.

PROCESS INTERRUPTION COMMANDS AND MACROS SUMMARY

Command/Macro	Function
BREAK Command	Temporarily pass control from a procedure or program to the control system.
EXIT Macro	Return from end-of-program interrupt routine to interruption point.

Command/Macro	Function
EXITP Macro	Permit control system to perform operations to allow reentry to program contingency routine.
INTR Command	Pass control to operators communication routine.
PAUSE Command	Temporily halt a task until operator performs some action.
RESUME Command	Resume object program execution or alter its order.
SETIC Macro	Simulate an interval timer interrupt.
SPEXT Macro	Obtain a Status Block which contains information required by user's interrupt routine.
STXIT Macro	Specify addresses for executive interrupt routine.

The following material describes each of the task and program management commands and macros. The descriptions include instruction format, programming considerations, and, where illustrative information is necessary, examples of the use of the instruction. The commands and macros are presented in alphabetical order without regard to functional classification.

ABEND Command

ABEND forces abnormal task termination. A PDUMP of Class 5 and Class 6 memory is taken and LOGOFF occurs automatically unless inhibited by the presence of the BUT operand. The operation of this command is effectively the save as CANCEL DUMP.

Name	Operation	Operand
	ABEND	[BUT]

BUT

Specifies that the terminal not be disconnected. The use of this operand causes the system to issue a PLEASE LOGON message to the user's terminal.

BREAK Command

BREAK allows the programmer to temporarily transfer control from either a procedure file or nonconversational program to the operating system. If the operating system is already in control when the BREAK command is received, the BREAK command is ignored.

Name	Operation	Operand
	BREAK	

Programming Notes

The BREAK command must be contained in either a nonconversational task or a PROCEDURE file specified in a conversational task. The BREAK command is read during program execution when SYSDTA is the same file as SYSCMD. A RESUME command must be issued to return control to the procedure file or nonconversational program.

CALL Macro

The CALL macro provides the programmer with a single statement entry for branching from a main program to subroutines contained within the same load program.

Name	Operation	Operand
	CALL	symbol

CANCEL Command

CANCEL is used to cancel a task. If the task to be canceled is waiting for task scheduling, it is eliminated from the task scheduling queue and any input parameters relating to it are bypassed.

If the task to be canceled is being executed, the devices and pages of storage reserved for its use are returned to the system.

If the task to be canceled has already been completed, the CANCEL command is rejected.

Name	Operation	Operand
	{CANCEL CAN}	tsn[,DUMP]

tsn

Specifies the task sequence number of the nonconversational task to be canceled. The number consists of a maximum of four decimal digits.

DUMP

Specifies that the program being run at the time the CANCEL command is received is to be dumped to the task's SYSLST file.

Programming Notes

A programmer cannot cancel a task identified by a LOGON associated task number. The programmer may only cancel tasks whose task numbers have been generated within a main task, such as ENTER files.

Users other than the System Controller may only cancel tasks generated under their userid as defined at LOGON time.

The CANCEL command can be issued only in the conversational mode.

The CANCEL command cannot cancel a spoolout task which is executing or in the spoolout queue.

The administrator will be prompted by the system to ensure that he wants to cancel the indicated task.

Remote Batch Processor task cannot be canceled.

CAM tasks cannot be canceled.

Example 1

/CAN 1234,DUMP

Task sequence number 1234 is canceled. The program running in task number 1234 when this command is issued is dumped to the SYSLST file.

Example 2

```
%C E222 PLEASE LOGON.
/LOGON USER-ID
%C E223 LOGON ACCEPTED AT 1024 ON 06/22/70, TSN-1392 ASSIGNED.
/EXEC (EDIT)
%L001 DYNAMIC LOADER INVOKED
 VERS. 0011 OF FILE EDITOR READY
*OPEN ISAM.FILE,OLD
 OPENED ISAM.FILE AS OLD V-TYPE FILE.
*TEXT
     3000.          <0>
*/LOGON CLARTEST
     3100.          <0>
*/PARAM LIST=YES,DEBUG=YES,ERRFIL=YES
     3200.          <0>
*/SYSFILE SYSDTA=DATA.FILE.SAM
     3300.          <0>
*/EXEC BGFOR
     3400.          <0>
*/SYSFILE SYSDTA=(PRIMARY)
     3500.          <0>
*/EXEC TSOSLNK
     3600.          <0>
*PROG NAME
     3700.          <0>
*INCLUDE *C_
     3800.          <0>
.*END
     3900.          <0>
*/LOGOFF
     4000.          <0>
*#END
*HALT
/ENTER SAM.FILE
 %D E172 TSN=1397.
/CANCEL 1397                    } ①
/CANCEL 1392                    } ②
%CANNOT CANCEL OWN TASK
/LOGOFF
%C E420 LOGOFF AT 1035 ON 06/22/70, FOR TSN 1392.
%C E421 CPU TIME USED: 004.9756 SECONDS.
```

Notes:

(1) User canceled task associated with the processing of SAM.FILE. TSN 1397 identifies the ENTER file task generated within the main task.

(2) User attempted to cancel the task identified by the primary task number. TSN 1392 identifies the programming session associated with the user's LOGON statement.

CHKPT Macro

The CHKPT (Checkpoint) macro is that half of the checkpoint restart combination that permits a programmer to specify node points at which system environment and program status are to be noted. These checkpoints can then be used, by way of RESTART command, to resume the execution of aborted programs.

Name	Operation	Operand
	CHKPT	fcb-addr,error-addr,restart-addr,
		id-char [,MF= {E,{(1) | addr} | L}]
	CHKPT	,MF=D

fcb-addr

The address of a P1FCB which has been used to open the PAM file to be used for checkpointing.

error-addr

An address in the user's program to which control is to be transferred if an error occurs during checkpoint.

restart-addr

An address in the user's program to which control is to be transferred after a restart, from the checkpoint, has been accomplished.

id-char

A 6-byte character string identifying the checkpoint.

MF=L

The inline parameter list (fcb-addr through id-char) is to be generated but not executed.

MF=E,(1)

The execution of an external parameter list, containing the same information as would have been supplied in the inline parameter list, the address of which is contained in register 1.

MF=E,addr

The execution of an external parameter list, containing the same information as would have been supplied in the in-line parameter list, the address of which is explicitly given.

MF=D

A DSECT is to be generated describing the parameter list by tagged items.

Upon being invoked, the checkpoint routine validates the parameter list supplied to it by the user. The routine then quiets any outstanding IO's and determines where in the file the checkpoint is to be written. The checkpoint comprises identification information, program status, related system status, and virtual memory contents all of which, when combined, provide the means of restarting the program logically at the checkpoint.

When the checkpoint has been successfully completed, a message is logged on SYSOUT for use at restart time. If errors prevent completion of the checkpoints, the user is given control at a specified error address and is passed a code specifying the nature of the error.

The primary direct output of the checkpoint process is a series of PAM pages of identifying information, necessary control blocks, information required for file reopening, and the contents of virtual memory. The checkpoint process also directs a message to SYSOUT when a successful checkpoint has been completed. This message associates the specified ID with a half-page number for subsequent use in restarting.

Programming Notes

The user can reduce the amount of time consumed by the checkpoint process by controlling the allocation of the file space for the checkpoint file. The approximate number of PAM pages used for a single checkpoint may be calculated using the formula

$$P=2n+6$$

where n is the number of pages of virtual memory allocated to the program at checkpoint time.

It follows, therefore, that the checkpoint should be taken at a point in the program where the amount of Class 5 and Class 6 memory allocated to the program is at a minimum.

Furthermore, the initial allocation of file space should be large enough to accommodate all anticipated checkpoints, thus avoiding secondary allocations. If this is not feasible, the next best alternative is to ensure that the secondary allocations equals or exceeds the requirements of any one checkpoint.

Cautions and Errors

If the checkpoint process is aborted, an error code is passed to the user in the low order byte of register 15. The register 15 codes are as follows.

X'00'	Checkpoint processed successfully.
X'04'	Unable to REQM Class 5 checkpoint workspace.
X'08'	Error in parameter list supplied by caller.
X'0C'	Checkpoint file not opened.
X'10'	No secondary allocation or illegal write.
X'14'	FCB not PAM, or OPEN not INOUT or OUTIN.
X'18'	Unrecoverable error returned by DMS.
X'1C'	Catalog Management error.

The preceding codes are in addition to the error exit byte and error code that may be in the FCB for codes X'0C', X'10', and X'18'. Control is returned to the caller's error address whenever an error is detected.

CSTAT Macro

The CSTAT (Change Status) macro instruction changes the status of a specified page or all of the pages owned by a Class II program. This macro also accepts the MF parameter.

Name	Operation	Operand
	CSTAT	PGNUM= {value / ALL} [,ACCESS= {READ / WRITE}] [,PAGE= {YES / NO}] [,MF= {L / (E,list) / (E,(1))}]

PGNUM=value

Specifies the page number of the page whose status is to be changed.

PGNUM=ALL

Specifies that all pages currently held by the program are to have their status changed.

ACCESS=READ

Specifies that the page is to be made read-only.

ACCESS=WRITE

Specifies that the page can be both read from and written to.

PAGE=YES

Specifies that the page is to be made pageable (nonresident).

PAGE=NO

Specifies that the page is to be made nonpageable (resident).

MF=

Refer to the explanation, "Type of Macro Instructions", in "Command/Macro Notation Conventions" of the Appendices.

Programming Notes

CSTAT cannot be used in a Class I program.

PGNUM=ALL may not be used when ACCESS parameter is specified.

When ACCESS parameter is omitted, no change is made to the read/write status of the page.

When PAGE parameter is omitted, no change is made to pageable status of the page.

Cautions and Errors

The General Register 15 Return Codes follow:

X'00'	Request was processed successfully.
X'04'	Page specified is not owned by the program.
X'0C'	An invalid request was encountered.
X'10'	The program has attempted to make more pages resident than were reserved for it at execution time.

DO Command

The DO command enables a programmer to execute a procedure (PROC) file. This command corresponds to the macro call in the assembly language. Following the PROC filename in the operand field of the DO command, the programmer may specify the operands which are to be inserted in place of the symbolic parameters. Positional and keyword operands may be mixed in the operand field of the DO command.

Name	Operation	Operand
	DO	filename[,(parameter-alphanum,...)]

filename

The filename of the cataloged file which is to become the temporary command input file (SYSCMD).

The filename is a set of up to 54 alphanumeric characters in the following form:

[$user-id.]name[.name...][(group)]

Userid is optional and consists of from one to eight characters, the first of which must be alphabetic.

Name is required. Extra .names can be specified.

The length of the name [.name...] field cannot exceed 44 characters.

parameters

Positionals and keywords which are to be inserted in place of the symbolic parameters in commands in the named procedure file. A comma must separate the positional or keyword operands from the file name. The positional or keyword operands are enclosed in parentheses and are separated by commas. If the default symbolic operands are desired, then the left and right parentheses must be used: for example, /DO filename, ( ).

Programming Notes

Positional operands must correspond to the positional symbolic parameters in the PROCEDURE statement. Keyword operands present as symbolic parameters in the PROCEDURE command may be omitted in the DO command, as default values can be specified. Additionally, any operand specified in the PROCEDURE statement may be omitted in the DO command. Refer to the PROCEDURE statement discussion for further information on procedure file definition.

See Section 2, Part 2, for the discussion of use and structure of procedure files.

Example

An example of the use of the DO command is contained in the procedure file discussion of Section 2, Part 2.

ENTER Command/Macro

The ENTER command provides the programmer with a method for initiating a nonconversational background task from a remote terminal or from within a nonconversational task. The nonconversational task is independent of the conversational task that issued the ENTER command and is identified by its own task sequence number.

Name	Operation	Operand
	{ENTER / E}	filename,[userid],[accountno],[password] [,PRIORITY=n] [,MSG={F / C} [H]] [,PRIVATE={YES / NO}] [,TIME=t]

filename

Specifies the name of cataloged file which will become the SYSCMD, SYSDTA, and SYSIPT of the nonconversational task. Filename must be a valid filename as described in the FILE command discussion. See Section 2 of this part.

userid

Specifies the user's identification and may consist of from one to eight alphanumeric characters. The first character must be alphabetic.

accountno

Specifies the account to be charged for the computer time used. The account number may consist of from one to eight alphanumeric characters.

password

May consist of from one to eight bytes written as either a character constant or a hexadecimal constant.

Examples: C'%( )*B2' and X'A0B0C0'.

The password consists of the information contained between the pair of single quotes.

PRIORITY=n

One digit in the range 1 to 9 specifying the priority for the task.

MSG=

Indicates how much of system messages is to be typed. F indicates that full system messages are desired. C indicates that only the code and variable portion of system messages are to be printed. If the MSG operand is not specified, the standard specified during system generation is used. H indicates that hard copy is desired and that all terminal input/output is to be written to the SYSLST file.

PRIVATE=

Specifies whether the ENTER file is on a private device (YES) or not (NO). NO is the default case.

TIME=

Specifies the number (a decimal integer) of minutes of CPU time allowed for this ENTER file.

Programming Notes

Entered files cannot be password protected. Therefore, RDPASS and WRPASS passwords cannot be honored. The task entered is independent of the conversational task in which the ENTER command was issued. The system assigns a task sequence number of the entered task and types this number on the user's terminal immediately after receiving the ENTER command.

The entered file must have as its first and last commands LOGON and LOGOFF, respectively. However, the optional parameters, userid, account number, priority, and message, when specified by the user in the ENTER command, override any of these parameters found in the file itself.

EOF Command

EOF is used within a program in conjunction with the RDATA macro to pass control to a program's end-of-file address.

EOF can also be used independent of the program to transfer out of a conversational task loop.

Name	Operation	Operand
	EOF	

Programming Notes

One of the operands of the RDATA macro is the end-of-file address. When the RDATA macro is executed, it requests a record from the SYSDTA file. (The device involved can be the user's terminal, a disc cataloged file, or a system card reader.) On normal reads, control is returned to the user's program at the location following the RDATA macro. When the record thus read is an EOF command, control is returned to the user's program at the end-of-file address.

To effect the transfer out of a conversational task loop, the programmer enters the EOF command at the terminal and then activates the terminal's Break Key.

ERFLG Macro

The ERFLG macro (Set Error Flag) is issued by system programs which wish to prohibit loading and executing of other programs within the step. This macro requests the TSOS Executive to set an error flag in the task's Monitor Table and Task Control Block (TCB) which prohibits loading and execution of programs encountered within the step. Control is returned to the user.

Name	Operation	Operand
	ERFLG	blank

Programming Notes

The ERFLG macro is provided for TOS compatibility only.

EXECUTE causes a program to be loaded into memory and to be executed starting at its initial entry point. EXECUTE can be issued either conversationally or nonconversationally.

Name	Operation	Operand
	{EXECUTE / EXEC}	{program / * / (module[,filename]) / Δ} [{,CLASSI = (max,min) / ,CLASSII = (max,min,p)}] [,TIME=t] [,IDA=YES]

program

Specifies the name of the cataloged file containing the program.

*

Specifies the current object module file (EAM file). The module must be Class II.

module

Specifies the name of the object module or an entry point in the object module. (The module name consists of one to eight characters and the module must be Class II.)

filename

Specifies the file name of an object module library which contains the specified module.

Δ

Specifies that the object module is within the SYSDTA input stream.

CLASS I=

Indicates that the parameters given are memory requirements for a CLASS I program. (This operand is only significant if the "program" option is used.)

(max,min)

Specifies the additional maximum and minimum resident pages requested by the user for his program; (max,min) are decimal integers in the range 0-254.

CLASS II=

Indicates that the parameters given are memory requirements for a Class II program. (This operand is only significant if the "program" option is used.)

(max,min,p)

Specifies the maximum and minimum resident pages requested for this program and, p, the total number of pages required. (max,min,p) are decimal integers in the range 1-254. If the core requirements are omitted, the default conditions are found in the header record for the program. This means the file must be opened so that the core requirement can be obtained.

TIME=

Indicates that an amount, t, of CPU time is specified for this program. If omitted, the system obtains this time from the system standard specified during system generation. When the program has exceeded its allotted time, the system notifies the user. The system then reinitializes the original allotted time.

t

The CPU time, in minutes, as a decimal integer ranging from one to the system standard maximum which is allotted to this program.

IDA=YES

Permits the use of the Interactive Debugging Aid for on-line monitoring of the execution of his program.

Programming Notes

If the program to be executed is a TOS program and the loading of TOS programs is currently being prohibited, the program will not be loaded, and the system will skip to the next STEP command or LOGOFF command in the task stream.

For further information on program loading, see the description of TSOS Loaders in the VMOS Service Routines Manual.

Examples

```
/EXECUTE TRANS99.JAN31-68, CLASSI=(5,1)          ①

/EXEC (FILE7.DATA),CLASSII=(5,2,15)              ②

/EXEC (Z1Y2X3.A9B8),TIME=25,CLASSI=(3,3)         ③

/EXEC *,TIME=15                                  ④
```

① The program contained in the cataloged file, TRANS99.JAN31-68, is to be loaded and executed. This CLASSI program needs a minimum of one resident page and can use up to five resident pages. The CPU time allowed is taken from the system standard.

② The program contained in the cataloged file, FILE7.DATA, is to be loaded and executed. This Class II program needs a minimum of two resident pages and can use up to five resident pages. The program needs a total of 15 pages of virtual storage. Maximum CPU time is taken from the system standard.

③ The program contained in the cataloged file, Z1Y2X3.A9B8, is to be loaded and executed, starting at its initial entry point. This Class I program needs 3 resident pages and is allowed to run for a total of 25 minutes of CPU time.

④ The current object module file (EAM file) is loaded and executed. This must be a Class II module and is allowed to run for a total of 15 minutes of CPU time.

EXCOM Macro

EXCOM is used to get data from the Common Data Area into a program. The Common Data Area is a 4096-byte area maintained by the Executive. This area is common to all Class I and Class II programs.

Name	Operation	Operand
	EXCOM	location,data[,length]

location

Is a decimal number indicating the location relative to the first byte of the Common Data Area from which the first byte of data is to be moved. The positions in the Common Data Area are counted from 1_{10}.

data

Is the symbolic address in the user's program into which the data from the Common Data Area is to be moved.

length

Is the decimal number of bytes to be moved. If length is omitted, the implied length associated with the symbolic tag of the receiving field specified in the data operand is used.

Examples

Example 1

```
Name        Operation    Operand
                .
                .
                .
            EXCOM        4,AREA,5
                .
                .
                .
AREA        DS           CL5
```

Example 2

```
Name        Operation    Operand
                .
                .
                .
            EXCOM        10,AREA
                .
                .
                .
AREA        DS           CL6
```

Note: Six bytes, starting with the 10th byte of the Common Data Area, are moved to location AREA in the user's program. Because the length operand is not stated, the implied length of the receiving field, AREA, is used.

EXECM Macro

EXECM (Execute User Programs) enables the programmer to have a program terminate itself and initiate the execution of another program.

Name	Operation	Operand
	EXECM	"{program / * / (module[,filename]) / Δ} [{,CLASSI=(max,min) / ,CLASSII=(max,min,p)}] [,TIME=minutes]"

program

Specifies the name of the cataloged file containing the program.

*

Specifies the current object module file (EAM file). The module must be a Class II program.

module

Specifies the name of the object module or an entry point in the object module. The specified module must be Class II.

filename

Specifies the filename of an object module library which contains the specified module.

Δ

Specifies that the object module is within the SYSDTA input.

CLASSI=(max,min)

Specifies the additional maximum and minimum resident pages requested by the user for his Class I program.

(max,min) are integers in the range $0_{10} - 254_{10}$.

This operand is only significant if the "program" option is used.

CLASSII=(max,min,p)

Specifies the maximum and minimum resident pages requested for the Class II program and, p, the total number of virtual memory pages required.

(max,min,p) are integers in the range $0_{10} - 254_{10}$.

This operand is only significant if the "program" option is used.

TIME=

The maximum CPU time, as an integer, allowed for this program. If omitted, this command will assign the allotted time which is obtained from the system standard (system generation). When the allotted time is exceeded, the operator is notified and the original allocated time is reinitialized.

Programming Notes

The entire operand field must be enclosed in quotes. After an EXECM has been issued, the issuing program cannot be reentered during the run.

EXIT Macro

EXIT (Exit from Contingency Routine) permits the programmer to supply information at the end of a program's interrupt routine that enables the executive to return control to the point in the program at which the interrupt occurred.

Name	Operation	Operand
	EXIT	{PR / TR / CR}

PR

Indicates return from the program's program check routine.

TR

Indicates return from the program's timer routine.

CR

Indicates return from the program's operator communication routine.

Programming Notes

Contingency routines which are terminated by the EXIT macro handle the following interrupts:

1. program check interrupt,

2. program timer interrupt, and

3. operator communication interrupt.

The macro expansion contains the SVC code associated with the referenced interrupt.

EXITP Macro

EXITP (Exit Provided by User) allows the programmer to supply information within a contingency routine that will allow the Executive to perform the necessary functions so that the contingency routine may be reentered on the next applicable interrupt.

Name	Operation	Operand
	EXITP	{ PR / TR / CR / UR / ER }

PR

Indicates return from the user's program check routine.

TR

Indicates return from the user's timer routine.

CR

Indicates return from the user's operator communication routine.

UR

Indicates return from the user's unrecoverable error routine.

ER

Indicates return from user's program-time-runout routine.

Programming Notes

It is the programmer's responsibility to restore General Registers 10 and 11, if so desired, by obtaining their contents with a SPEXT macro in a Class II program or with an ADEXT macro in a Class I program.

After execution of this macro, the Executive returns control to the instruction following the macro expansion.

GETSW Macro

GETSW (Get Switch) permits the programmer to store in General Register 0 the settings of the 32 task switches for the task containing the program in which the GETSW is issued.

Name	Operation	Operand
	GETSW	

Programming Notes

The task switches are 32 bits maintained (for each task) by the Executive. The 32 bits are initially set to 0. Through the use of the SETSW (Set Switch) macro, the user can selectively set, and optionally reset, the bits to a value of 1.

Bit 2^0 of General Register 0 corresponds to task switch 0, bit 2^1 corresponds to task switch 1, and so on.

The task switches are a mechanism of communication from job to job within a task. For example, a program might communicate to a successor program the presence or absence of an output file which is optional input to the successor program.

INTR Command

The INTR (interrupt) command permits the programmer to cause control to be transferred from the program to the operator communication routine as specified in the program's STXIT macro.

Name	Operation	Operand
	INTR	{ blank / tsn }

blank

The operand field is left blank when the INTR command is used in the conversational mode.

tsn

When the INTR command is used in the nonconversational mode, the task sequence number of the task is given as the operand.

Programming Notes

To pass control from conversational tasks, the programmer interrupts the operation of the program by pressing the BREAK key, then types the INTR command.

The passing of control from a nonconversational task requires operator intervention as described in the VMOS Operations Management Manual.

The LOAD command enables a programmer to specify that an object module be loaded into memory. The execution of the program is not begun, however, until the user explicitly indicates so (see RESUME command). Subsequent to the LOAD command and prior to the call for execution, the programmer may enter other commands without affecting the status of the loaded program.

Name	Operation	Operand
	LOAD	{ program | * | (module[,filename]) | Δ } [{ ,CLASS I = (max,min) | ,CLASS II = (max,min,p) }] [,TIME=t] [,IDA=YES]

program

Specifies the name of the cataloged file containing the program.

*

Specifies the current object module file (EAM). The module must be in Class II.

module

Specifies the name of the object module or an entry point in the object module. The module name consists of one to eight characters and the module must be in Class II.

filename

Specifies the filename of an object module library which contains the specified module.

Δ

Indicates that the object module is within the SYSDTA Input.

CLASS I

Indicates that the parameters given are memory requirements for a Class I program. (This operand is only significant if the "PROGRAM" option is used.)

CLASS II

Indicates that the parameters given are memory requirements for a Class II program. (This operand is significant only if the "PROGRAM" option is used.)

(max,min,p)

Specifies the maximum and minimum resident pages requested for this program and, p, the total number of pages required. (max,min,p) are decimal integers in the range 0-254.

TIME=

Indicates that an amount, t, of CPU time is specified for this program. If omitted, the system obtains this time from the system standard specified during system generation. When the program has exceeded its allotted time, the system notifies the user. The system then reinitializes the original allotted time.

t

The CPU time, in minutes, a decimal integer ranging from one to the system standard maximum.

IDA=YES

Permits the use of the Interactive Debugging Aid for on-line debugging.

Note: If memory requirements are omitted (the Class I and Class II parameters) in this macro, the system expects to find the memory requirements in the header record of the program.

Programming Notes

If the program to be executed is a TOS program and the loading of TOS programs is currently being prohibited, the program will not be loaded, and the system will skip to the next STEP command or LOGOFF command in the task stream.

For further information on program loading, see the description of TSOS Loaders in the VMOS Service Routines Manual.

Examples

```
/LOAD PROG1                                          (1)

/LOAD (FILE.NAME),TIME=10,CLASSII=(6,4,6)            (2)

/LOAD (FILENAME.A),CLASSI=(10,1),TIME=20             (3)
```

① The program contained in the cataloged file, PROG1, is loaded. Execution is not begun until a RESUME command is issued. Memory requirements for this program are taken from the file's header record. Allowed CPU time for this program is taken from the system standard.

② The program contained in the cataloged file, FILE.NAME, is loaded. Execution is delayed. The program is authorized to use 10 minutes CPU time. This ClassII program needs a total of six pages of virtual storage, four of which must be resident.

③ The program contained in the cataloged file, FILENAME.A, is loaded. This ClassI program is allowed to use 20 minutes of CPU time and requires a minimum of one resident page. Up to 10 pages can be made resident for this program.

LOADM Macro

LOADM macro enables the program to terminate itself and to load a new user program, but control is passed to the SYSCMD file instead of the called program. The calling program is no longer available.

Name	Operation	Operand
	LOADM	"{program / * / (module [,filename]) / Δ} [{,CLASSI=(max,min) / ,CLASSII=(max,min,p)}] [,TIME=minutes]"

program

Specifies the name of the cataloged file containing the program.

*

Specifies the current object module file (EAM). The module must be Class II.

module

Specifies the name of the object module or any entry point in the object module. The module name consists of one to eight characters and the module must be Class II.

filename

Specifies the filename of an object module library which contains the specified module.

Δ

Indicates that the object module is within the SYSDTA input.

CLASS I=(max,min)

Specifies the additional maximum and minimum resident pages requested by the user for his Class I program.

(max,min) are integers in the range $0_{10} - 254_{10}$

This operand is only significant if the "program" option is used.

CLASS II=(max,min,p)

Specifies the maximum and minimum resident pages requested for the Class II program, and, p, the total number of virtual memory pages required.

(max,min,p) are integers in the range $0_{10} - 254_{10}$.

This operand is only significant if the "program" option is used.

TIME=

The maximum CPU time, as an integer, allowed for this program. If omitted, the command will assign the allotted time. The allotted time is obtained from the system standard (system generation). When the allotted time is exceeded, the operator is notified and the original allotted time is reinitialized.

Cautions and Errors

1. The entire operand must be enclosed in quotes.

2. If memory requirements are omitted (the Class I and Class II parameters) in this macro, the system expects to find the memory requirements in the header record of the program.

LOGOFF Command/Macro

The user ends a task with a LOGOFF command; therefore, LOGOFF must be the last command in every task. LOGOFF causes the TSOS Executive to remove the task from the system and return to the system the virtual storage and any input/output devices used by the task. The BUT option allows the conversational user to LOGON again without the terminal being disconnected.

Name	Operation	Operand
	LOGOFF	[{BUT[,TAPE] / TAPE[,BUT]}]

BUT

This operand pertains only to conversational users and is ignored in the nonconversational mode. When employed, the conversational user will be requested to LOGON after present task has been destroyed. When omitted, the conversational user's terminal is disconnected prior to task destruction.

TAPE

This operand requests the system files to be spooled out to tape.

The SYSLST and/or SYSOUT files will be written to the same tape. The filename will be TAPE.TSNnnnn, where nnnn is the task sequence number of the task which created these files.

The SYSOPT file will be written to a separate tape. It will also have a filename of TAPE.TSNnnnn, but the nnnn will be a new task sequence number which is assigned to the task which will spool out this file to tape. The user will be notified of this TSN in his SYSOUT file.

Programming Notes

All files which are spooled out to tape are cataloged SAM files. They are variable length records and the first data character is a machine code control character. The maximum block size is 2064 bytes. The first 16 bytes contain the key. The remaining 2048 bytes are a record in the standard variable format.

The file may be printed by using a PRINT command with the SPACE=E option, or punched via a PUNCH command with the STARTNO=2 option, to by pass the control character.

The LOGON command identifies the user to the system. A user cannot communicate with the system until he has successfully logged on and his user's credentials (id and password) are verified.

The sever bit in the JOIN table entry is checked to ensure that this user can LOGON. Only the userid operand is mandatory. If no account number or an incorrect account number is provided in the LOGON command statement, the first one in the JOIN table entry is used. If a password is in the JOIN table, yet is omitted or incorrect in the LOGON command, the user in the conversational mode is not logged on, the user in the nonconversational mode is aborted. In general for illegal operands, the conversational user is prompted, the nonconversational user is aborted.

Name	Operation	Operand
	LOGON	userid,[accountno], [password][,PRIORITY=n] [,MSG{F / C / FH / CH}] [,BUFFER=integer] [,PRIVATE={YES / NO}] [,TIME=sec]

userid

Is from one to eight alphanumeric characters (the first of which must be alphabetic) used to identify the user to the system. This is the only mandatory operand in the LOGON command. Imbedded blanks in the userid are not allowed.

accountno

From one to eight alphanumeric characters (no blanks) indicating the account to be charged for the computer time used. If no account number is specified, the first one in the Join Table entry is used. If an account number is specified but is not in the Join Table entry, the system types a prompting message.

password

From one to eight characters specifying the user password stated in character or hexadecimal constant format. C'%()*B2' and X'A0B0C0' are two valid examples of passwords. Terminal control characters cannot be used for password characters.

If the Join Table entry for this user contains a password and the LOGON operand contains a different password (or no password), the system types a prompting message.

PRIORITY=

A 1-digit numeric code in the range 1-9 which specifies the priority for this task (1 is highest priority, 9 is lowest); n cannot exceed the priority indicated in the Join Table entry. If PRIORITY= is omitted, the installation standard specified at system generation is used.

MSC=

Controls the length of a systems message to be typed. A 1-character code, F or C must be specified. F indicates that full system messages are to be typed. C indicates that only the code and variable portion of a system message are to be printed. If the MSG operand is not specified, the installation standard is used.

FH and CH permit the user to specify the hard copy option. When the user exercises this option, both the system message reads/writes and user message writes/reads are written to the SYSLST file which is spooled out when the user logs off. The FH and CH entries indicate the same message lengths and formats as F and C, respectively.

BUFFER=

Allows the user to request an arbitrarily large terminal input buffer. Minimum buffer size is 80 bytes. Maximum buffer size is 1024 bytes. If this operand is omitted, a buffer of 80 bytes is allocated.

PRIVATE=

YES indicates that the user requires private devices. NO indicates that the user does not require any private devices. The default condition is YES (for private devices).

If the user has indicated he requires no private devices, but in the subsequent execution of the job stream requires a private device, the task will be abnormally terminated. This operand is only significant for nonconversational tasks.

TIME=

Requirement in minutes for CPU time. If omitted, the time is obtained from the system standard (system generation). This operand is only significant for nonconversational tasks. If the value given or defaulted is greater than that amount remaining for the account, no LOGON is permitted. If the value is greater than/BIAS time limit, the job remains on queue.

Programming Notes

The password in the nonconversational LOGON statement will be cleared to spaces before the record is written to SYSOUT.

Examples

Example 1

```
%C E222 PLEASE LOGON
/LOGON DDD
%C E223 LOGON ACCEPTED AT 0930 ON 12/23/69, TSN
0876 ASSIGNED
```

Example 2

```
%C E222 PLEASE LOGON
/LOGON DDD,, PRIORITY=3,MSG=C,BUFFER=500
%C E223 LOGON ACCEPTED AT 0938 ON 12/23/69,TSN
0887 ASSIGNED
```

The userid of DDD is submitted for validation. A priority of 3 is specified.

A coded typeout of all system messages is requested and an input buffer size of 500 characters is requested.

LPOV Macro

LPOV (Load Program Overlay) causes a load module to be loaded during execution of the user program.

Name	Operation	Operand
	LPOV	module [,address]

module

Specifies the symbolic name of the module to be loaded. This symbol can be 1-6 alphanumeric characters.

address

Specifies the symbolic name of the location in the user program to which control is given after the module has been loaded. If this operand is omitted, control is given to the instruction following the macro expansion.

The program is waited until the module is loaded.

Programming Notes

When errors occur during loading of a Class I overlay load module, the Executive terminates the task.

When errors occur during loading of a Class II overlay load module, the Executive does the following:

1. places an error code (listed below) in P1 GR 15.

2. returns control to the user program at the instruction following the LPOV macro expansion.

Cautions and Errors

General Register 15 Return Codes:

X'00'	The load was successful.
X'04'	The text record that was read extends over a page boundary, or there is a problem in the linkage editor, or the loader routine cannot be loaded.
X'08'	The format of the text modifier index record for this load module is illegal.
X'0C'	The name in the load module index record does not correspond to the name in the text modifier index record.
X'10'	The requested load module cannot be found.
X'14'	There is insufficient memory available to load this module.
X'18'	An error occurred when reading a record from a random access device.

PASS Macro

PASS causes the task to relinquish the remainder of its current time slice. The task is rescheduled in the next cycle, and its times slice is reinitialized.

Name	Operation	Operand
	PASS	

Programming Notes

The PASS macro is commonly issued by a CAM program when there is no line activity.

PASS can be issued by a task when requested system facilities are not available.

PAUSE Command

The PAUSE command causes the task (either conversational or nonconversational) to be temporarily halted until the operator has performed some action. A message is sent to the operator's console and the operator resumes execution of the task by responding to the message.

Name	Operation	Operand
	PAUSE	message

message

specifies the message to be typed on the operator's console typewriter. The message cannot exceed 72 characters.

Examples

```
/PAUSE OPERATOR ---- MOUNT PACK #538    ①

/PAUSE PUT CARD DECK IN READER    ②
```

① The message "OPERATOR....MOUNT PACK #538" is typed on the operator's console typewriter. When the operator has done so, he responds to the message. Control is then returned to the task.

② The message "PUT CARD DECK IN READER" is typed on the operator's typewriter. When the operator has readied the card reader, he responds to the message. Control is then returned to the user task.

PROUT Macro

PROUT (Write Print Line Image) causes a record to be written to the SYSLST file.

Name	Operation	Operand
	PROUT	address[,length]

address

Specifies the symbolic address in the user's program of the record to be written to SYSLST.

The first character must be a print control character.

length

Specifies the length of the output message, including the printer control character. This record is in fixed length format. If the length parameter is omitted, the length attribute of the area (including the control byte) is taken to be the length of the record.

The Executive converts the user's records to V-type records prior to writing them to SYSLST. Unless a WRLST was previously issued, a SYSLST EAM file is created by the Executive when the first PROUT for a task is issued. With the exception of those instances when the output is destined for a remote station, SYSLST is spooled out onto a line printer at job termination. After spooling, the SYSLST file is erased.

If a nonrecoverable error occurs during execution of the PROUT macro, the program is terminated, and control is given to the next STEP or LOGOFF command in the command input stream.

Programming Notes

The print control character is one byte of control information specifying the type of paper advance desired when SYSLST is spooled out to the printer.

RDATA Macro

RDATA (Read Data from SYSDTA) enables the user program to retrieve the next data record from the SYSDTA file. The device associated with the SYSDTA file can be the user's terminal, a system card reader, or (a cataloged file on) a disc. The user designates the area in his program into which the system places the record. The System enters the record as a variable format record.

The user can formulate his own parameter area and lead its address into General Register 1. In this case, he uses the second format of the macro instruction.

Name	Operation	Operand
	RDATA	inarea,erraddr[,length] [,edit] [,A] [,KEYOUT= {Y / N}] [,MF=] [,EOF=end-of-file address] [,KEYPOS= {Y / N}] [,KEYLEN= {Y / N}]

RDATA (1)

inarea

Specifies the symbolic address in the user program into which the next record from SYSDTA is to be read.

erraddr

Specifies the address in the user's program to which control is given when an error occurs or an assignment change is made (including initial assignments) and notification was requested.

length

Specifies the size of the user's "inarea." This area must be four bytes larger than the largest record expected because V-type format records require an initial 4-byte field which indicates the actual number of bytes read.

If the length parameter is omitted, the length attribute of "inarea" is assumed to be the length of the record to be read.

edit

Specifies the editing option for a record to be read from the user's terminal, or sent to the user's terminal. The edit parameter is required only when a nonstandard translation is required from the terminal. The options are as follows:

x'00'

Translation from ASCII to EBCDIC and deletion of line feed and carriage return characters from the text.

x'04'

SYSDTA is a sequential file.

x'08'

SYSDTA is an indexed-sequential file.

x'0C'

SYSDTA received from a card reader in the central computing center.

x'10'

SYSDTA received from a remote card reader (e.g., 8741).

x'14'

SYSDTA received from a terminal.

KEYOUT= Y

Indicates a desire to have the ISAM KEY stripped from the record being processed before the record is passed to the user's specified read area. The Flag Table byte in the parameter table is set to x'80.'

N

Indicates that the ISAM KEYS are not to be stripped from records. This is the default case.

MF=

Indicates a desire to have the RDATA macro expanded according to the system standard conventions defined for this option. Refer to "Command/Macro Notation Conventions" in the appendices.

EOF = end-of-file address

Causes the Flag Table byte to be set to x '40' and control to be passed to the indicated address when an EOF, End-of-File, is processed. The user is cautioned that this option generates an extra word at the end of the RDATA parameter list. If this option is not specified, an EOF causes control to be passed to the user's error exit.

KEYPOS= Y

Causes the Flag Table byte to be set to x '20' and the ISAM KEY position value to be returned in the right hand portion of the user's register zero.

N

The Flag Table byte is not set to x'20' and the ISAM KEY position value is not entered in register zero. This is the default option.

KEYLEN= Y

Causes the Flag Table byte to be set to x '10' and the ISAM KEY LENGTH value to be returned to the right-hand portion of the user's register zero.

N

The Flag Table byte is not set to x'10' and the ISAM KEY LENGTH is not entered in register zero. This is the default option.

(1)

Indicates that the operands are found in a parameter table whose address has been loaded in register 1.

The Control Program places the parameters specified in the operand in a parameter area and loads the starting address of this area into the General Register.

The contents of the parameter area are as follows:

Item	Byte	Form
Edit Option	0	Binary
Address of read input area	1-3	Binary
Reserved	4-5	Binary
Length of read	6-7	Binary
Assignment change indicator	8	Binary
Address of user's error routine	9-11	Binary
Reserved	12	Binary
End-of-File Address (optional word generated for EOF parameter)	13-15	Binary

If the assignment option is requested, an x'01' is set in the assignment change indicator in the parameter table. When the assignment option is omitted, an x'00' is set in the indicator.

Programming Notes

If the user has specified the "A" option and receives control at his "erraddr," General Register 15 contains one or both of the following:

1. Assignment code in left-hand byte.

2. Error code in right-hand byte.

General Register 14 will contain the address of the instruction in the user program following the macro expansion.

When issued during a DXC session, the RDATA macro will cause an MT=6 Message Type to be transmitted to the auxiliary processor. MT=6 causes the auxiliary processor to write an asterisk on a new line at the user's terminal and be prepared to read a record from the terminal.

If both the KEYPOS and KEYLEN parameters are used, the right-hand byte of register zero contains the key length and the next two bytes following it contain the key position.

Cautions and Errors

General Register 15 Error Return Codes:

x'00'	Normal termination
x'04'	Nonrecoverable error
x'08'	Parameter error
x'0C'	Read truncation
x'10'	End of file

A nonrecoverable error is a consistent hardware malfunction. A parameter error is an error in one of the user specified operands. Read truncation is an error in which the record being read is larger than was specified in the length operand of the macro instruction. The actual record is truncated to fit in the size specified, minus four bytes for the length field. (If the record is shorter than specified, the system places it in the area left justified with no space fill and executes a normal return to the user.) The end of file condition, though not an error condition, causes control to be transferred to the user's error address. The end of file condition can occur in the following ways:

terminal

By pressing the break key when a read is requested and keying in the EOF and RESUME commands.

other devices

(1) When the SYSDTA file and the SYSCMD file are the same and a command (other than the BREAK command) is read.

(2) When the SYSDTA file and the SYSCMD file are not the same and a /EOF is detected in columns 1-4 of the record.

Example

```
Name        Operation    Operand
            LA           1,AREA
            RDATA        (1)
            .
            .
            .
            .
AREA        DS           CL12
```

Note: The user loads the address of his parameter area into General Register 1 before issuing the RDATA macro.

The parameters have been stored at location AREA according to the format outlined in the introductory description.

RDCRD Macro

RDCRD (Read Record from SYSIPT) instructs the Executive to get an input record from the SYSIPT file. The SYSIPT file can be a cataloged data file or a system card reader.

Name	Operation	Operand
	RDCRD	area,address

area

Specifies the symbolic address in the user's program of the area into which the input from SYSIPT is to be placed. This record is in fixed length format.

address

Specifies the symbolic address in the user's program to which control is given when the end of file condition is detected. A command statement (record with / in column 1) causes an end of file condition except when SYSIPT is the same as SYSCMD and a /BREAK command is read.

Programming Notes

All data records thus read from SYSIPT are 80 bytes in length. Longer records are truncated to 80 bytes and shorter records are blank-$(40)_{16}$-filled to make an 80-byte record. The records read are placed in a location designated by the user. An end-of-file condition is caused by a command statement being read from SYSIPT. Then control is passed to the user's end of file address specified in the macro operand.

When SYSIPT is an ISAM File, an 80-byte record is provided to the requestor. The 8-byte key is placed into positions 73 and 80 of the record. If records are less than 72 bytes long, spaces are placed from the end of data up to the 73rd position of the record.

Data records are placed in the user designated area and control is returned to the user at the location following the macro expansion.

In Class I programs, if a RDCRD to a cataloged file results in a /* record, the record is passed to the user program.

Command statements cause an end of file condition and the user receives control at his end of file indicated address.

A nonrecoverable error causes the program to be terminated. Subsequent commands are ignored until a STEP or LOGOFF is encountered.

REMARK Command

REMARK allows the user to output remarks to the SYSOUT file. For a conversational user who is not in the PROC mode, remarks are not written.

Remarks cannot be contained on multiple lines, but any number of REMARK commands can be issued.

Name	Operation	Operand
	REMARK	remarks

remarks

Any desired remarks can be entered. A remark should not exceed 72 characters in length.

Programming Notes

If the REMARK command is used in a procedure file, the command print option in the PROCEDURE command must be exercised for the remark to be written to the SYSOUT file.

Examples

```
/REMARK THIS GOES TO SYSOUT
```

The remark "THIS GOES TO SYSOUT" is output to the SYSOUT file. If SYSOUT is the user's terminal, the remark is not written.

```
/.NAME REMARK REMARKS
```

This command has a name field. The remark "REMARKS" is output to the SYSOUT file.

RESUME Command

The RESUME command resumes object program execution or to alter the normal order of object execution.

Name	Operation	Operand
	{RESUME / R}	[instloc]

instloc

Specifies the instruction location at which execution is to be started.

Default:

Execution resumes at the point the object module was interrupted.

Programming Notes

The operand of the RESUME command may be a hexadecimal instruction location, a register with the indirect indicator set, or a symbol with or without an offset. The instruction location must be on a halfword boundary. If no errors are detected, control is transferred to the object program at the specified instruction location.

The general rule concerning RESUME and STOP commands is the following: When these appear in AT statements they are always executed when the AT statement becomes effective; at all other times they are executed immediately or when a qualifying IF command calls for their execution.

RESTART Command

The RESTART command causes a program interrupted by a checkpoint (via the CHKPT macro) to be reloaded from the point at which the checkpoint was taken. Unless suppressed by the LOAD option, RESTART will also begin reexecution of the program at that point. The information required to restart the program can be specified when the RESTART is given by the user or obtained by the system from a log created when the checkpoint occurred. The RESTART command may be issued interactively (from the user's terminal) or in the background as a part of a spooled in job stream. However, this command cannot be issued from the operator's console.

Name	Operation	Operand
	RESTART	filename,trailerpage[,LOAD]

filename

The ID of the file from which the checkpoint records are to be taken.

trailerpage

The half-page on which the checkpoint is delimited. It is the last record of the checkpoint and contains the information necessary to reestablish the system and files as of that checkpoint.

LOAD

The means of indicating that control is to be returned to SYSCMD after the program has been reloaded, and execution is to be suppressed.

Programming Notes

The task under which restart is put into effect will reacquire the same user memory which the checkpointed task had when the checkpoint went into effect. However, the task will operate in the mode (nonconversational or conversational) as that from which RESTART was initiated.

The LOAD option provides the user with the same advantages that the LOAD command affords for program initiation, for example, the use of IDA to correct a program error that made restarting necessary.

The output, or result, of the restart process is the original user program in the same status as it was when the checkpoint took place.

RETRN Macro

RETRN (Return to a Program), when issued in a called program, returns control from a "called" program to a "calling" program if the called program follows the standard linkage conventions. In addition, it can be used to restore the contents of any General Registers which have been saved for the calling program by the called program.

The called program may use the SAVE routine to save the calling program's General Registers. The calling program must have previously defined a "save" area into which the registers may be stored.

Thus, the SAVE and RETRN macros provide a standardized method of establishing linkage into and out of a subroutine.

Name	Operation	Operand
	RETRN	$(r_1 [,r_2])[,T] \left[,RC=\left\{\begin{matrix} rc \\ (15) \end{matrix}\right\}\right]$

r_1 , r_2

Specifies one or a range of registers whose contents are to be restored from the save area of a calling program.

The r_n are decimal integers in the range 0-12, 14, and 15. Register 13 is not stored but is used by the calling program to transmit the address of the save area. Register 13 must be preserved by the called program.

The entire range of registers can be specified by the notation (14, 12). This will store registers 14, 15, and 0 through 12.

The save area is a 72-byte area (18 words). Words 4 and 5 of the save area are used to store the contents of Registers 14 and 15. Words 6 through 18 are used to store the contents of registers 0 through 12.

T

Specifies that a (1) is set in the low-order bit of the forward link, word 3, in the save area defined by the calling program. This action occurs after completion of the register reloading specified by the first operand. The bit is set to stop the forward chain.

This parameter is supplied to facilitate tracing (i.e., checking program flow). There is no training in TSOS; this parameter is provided for TOS compatibility with language translators (that is, when DEBUG = YES).

RC = rc

A code to be placed in the 12 rightmost bits of General Register 15. rc must be a decimal integer which is a multiple of 4 and in the range 0-4092. The rc operand can also be an absolute expression as defined in the Assembly Reference Manual.

RC = (15)

Indicates that the user has loaded the return code into the 12 rightmost bits of General Register 15 before executing the RETRN macro.

Note: The return code is used by agreement between the calling program and called program to communicate events occurring during the execution of the called program.

Programming Notes

The expansion of the RETRN macro does not contain a Supervisor Call but utilizes an LM instruction to restore specified registers.

The user (in called program) must either load General Register 14 correctly before issuing the RETRN or he must restore the contents of of General Register 14 by use of the first operand of the RETRN macro. (Register 14 is the "return register" containing the return address to the user's calling program.)

Example

```
Name        Operation   Operand
                .
                .
                .
            LA          13,AREA     } Calling
            BAL         14,SUBRTN   } Program
                .
                .
                .
AREA        DS          18F
                .
                .
                .
SUBRTN      SAVE        (14,12)
```

RJOB Command

The RJOB command is used to identify jobs submitted to the RBP system. The command marks the beginning of a job entry and is in effect only for the job which immediately succeeds it. The user has the capability to defer job output until it is requested by command and to designate an alternate output recipient.

The user submitting the job is notified of its entry into the system after the Spoolin process is complete. When the job has finished executing, the user and alternate output recipient, if one exists, are automatically notified of job completion. Deferred job output may be retrieved any time after receipt of the job completion message.

This command may be used to define processing for jobs submitted from a central installation card reader. This provides the capability of directing output from centrally submitted jobs to remote users.

RJOB is an optional command. If it is omitted, the system will supply a unique jobname and will direct job output to the submitting user immediately after job completion.

Name	Operation	Operand
	RJOB	jobname-alphanum
		[,{CENTRAL / DEFER[,userid-alphanum]}]

jobname

Identifies the job entry to the system. This allows the user to request and receive job output and job information by jobname. RBP operation requires individual jobnames. A subsequent job will be rejected if it is submitted with the name of a job currently in the system. The jobname consists of one to eight alphanumeric characters.

CENTRAL

Specifies that job output is to be directed to a central installation line printer immediately after job completion. The job output will not be available to the remote user submitting the job.

DEFER

Specifies deferred remote job output. The user must retrieve the output using the ROUT command. This parameter applies only to remote work station output and has no effect on output destined for a central line printer.

If this parameter is missing, the output will be sent to the work station submitting the job immediately after job completion. If this work station is inactive, the output is held until either the work station is logically attached or the user logs on at another work station. In this case, REP advises the central operator that output is available for an inactive work station.

DEFER,userid

Specifies that job output is deferred and that another remote user, identified by the userid, is a valid job output recipient. Only one copy of the job output is available and REP returns it to the first valid recipient requesting it. The userid parameter can only be used with the DEFER option.

RLOGOFF Command

With the RLOGOFF command, the user indicates that he has completed his session. REP rejects input from the work station until another RLOGON command is submitted. However, the work station can continue to monitor the system for output, or the user can issue an RSTOP command to detach the station from the system.

If the central system receives a valid RLOGON command from a work station with a session in progress, the central system logs off the current user and logs on the new user. If the central system receives an RSTOP command from a work station with a session in progress, it logs off the user and logically detaches the work station.

Name	Operation	Operand
	RLOGOFF	blank

RLOGON Command

The RLOGON command identifies a user to RBP. The system validates the userid, account number, and password and, if accepted, grants access to the system. At this point, the user can begin requesting use of RBP facilities.

The RLOGON command remains in effect until another RLOGON, an RLOGOFF, or an RSTOP command is issued.

A user cannot be logged on at more than one RBP work station at a time. If a user desires to change work stations, he must log off at his old work station before logging on at the new work station.

Name	Operation	Operand
	RLOGON	(standard LOGON operands)

Only the userid, accountno, password, and priority operands are used in the RBP logon process. The msg and buffer operands are ignored.

RMSG Command

The RMSG command allows a user to send a message to another user, the central operator, or a work station attached to the RBP system. Messages are rejected if they are directed to inactive user, or if they are directed to work stations that are not attached to the system. An active user is one who is currently logged on, or was the last user logged on at a currently attached work station. If the user specifies that a message is to be sent to both a user and a work station, RBP first tries to send the message to the user. If he is not active, the system tries to send the message to the specified work station. The message will be rejected if it cannot be sent to either one.

Name	Operation	Operand
	RMSG	M=C'text' [,U=userid-alphanum] [,T=termid-alphanum]

M=C'text'

Specifies the message text to be sent. The text must be framed by apostrophes and is limited to a maximum of 40 printable characters and blanks.

U=userid

Specifies the user that is to receive the message. The message is rejected if the user is not active. A userid of TSOS.OPR will direct the message to the central operator console.

T=termid

Specifies a work station that is to receive the message. The message is rejected if the work station is not currently attached to the system. A termid of TSOS.CTR will direct the message to the central operator console.

ROUT Command

The ROUT command allows the remote user to retrieve deferred job output for which he is a valid recipient. If the job is not complete when the command is processed, RBP returns a message indicating this, and the command must be resubmitted after the job has completed. The RBP system returns an invalid request response if the user is not a valid recipient or if the job is not in the system. A user is automatically a valid recipient of job output if he submitted the job.

This command also allows the user to retrieve output that was discontinued due to user intervention or equipment failure. When interrupted output is pending for a work station, no output is returned to the work station until an ROUT command determines the disposition of the interrupted job. The system continues to accept input from the work station, however. If the user submits an RSTOP command, transmission of the discontinued output will resume from the point it was discontinued, when the user submits his next RSTART command. If the discontinue state was the result of a transmission failure, the user must reattach his work station with an RSTART command.

Additionally, the ROUT command allows the user to delete jobs from the system without receiving a copy of the job output. The user can delete only jobs which he submitted and which are currently in the system. The RBP system returns an invalid request response if the user attempts to delete jobs that do not belong to him. Jobs may be deleted anytime after they are submitted. The ROUT command has no effect if the job output has already been returned since the system would have deleted all reference to the job.

Name	Operation	Operand
	ROUT	[{J=jobname=alphanum / U=userid-alphanum}, {CONTINUE / BEGIN / DELETE}]
	ROUT	U=ALL [, {CONTINUE / BEGIN}]

J=jobname

Indicates that the request is for the specific job named in the parameter.

U=userid

Indicates that the request pertains only to the jobs submitted by the named user. If the user gives his own userid, he receives or deletes all output from jobs submitted by the named user which designate the requesting user as a valid recipient.

U=ALL

Indicates that the request is for all output in the system for which the user is a valid recipient. The user receives all available output of jobs submitted by him and of jobs submitted by other users which named him as a valid recipient.

CONTINUE

Specifies that job output is to resume with the record that was being written when the output was discontinued. If job output is not in a discontinued state, this parameter has no effect since output will begin with the first record in the output file.

BEGIN

Specifies that job output is to begin with the first record in the output file, even if the output is in a discontinued state.

DELETE

Specifies:

1. That the named job (J=jobname) is to be deleted from the system. The job must belong to the user who is currently logged on at the work station.

2. That all current jobs submitted by the named user (U=userid) are to be deleted from the system. The only valid user is the one who is currently logged on at the work station. The system returns a message which specifies the name of each deleted job.

When both parameters are omitted, the system will resume output for a job that is in a discontinued state. If job output is not in a discontinued state, this form of the command merely resets the work station output device status to an enabled state.

RSTATUS Command

The RSTATUS command allows a user to determine the status of one or more remotely submitted jobs. RBP returns the status of only current jobs for which the requestor is a valid recipient. The user may request status for a specific job, for jobs submitted by a specific user for which he is a valid recipient, or for all current jobs that were submitted by this work station. RBP returns the status of only those jobs that are in the system at the time the command is processed.

The user receives a response for each job that satisfies the command. Each response contains the jobname, submitting user-id, submitting termid, alternate userid, and one of the following indications:

1. Job is executing.

2. Job is complete – output available – normal or abnormal termination.

3. Job is complete – output reserved by alternate or submitter – normal or abnormal termination.

4. Job is not in the system.

5. Invalid request (the user is not a valid recipient).

Name	Operation	Operand
	RSTATUS	{ J=jobname-alphanum U=userid-alphanum T=termid-alphanum }

J=jobname

Indicates that the request is for the specified jobname. If the user issuing the request is not a valid recipient, the request is denied.

U=userid

Indicates that the request is for the status of all current jobs submitted by the user identified with the userid for which the requesting user is a valid recipient. If the user specifies his own userid, he receives the status of all jobs in the system for which he is a valid recipient.

T=termid

Indicates that the request is for all current jobs in the system that were submitted from this work station. A report is made for all jobs in this category regardless of the submitting user.

SAVE Macro

SAVE permits the user to store the contents of specified general registers. SAVE is normally written at the entry point of a called program. The save area is defined by the calling program. The SAVE macro in conjunction with the RETRN macro provides a means for a called program to save the register contents of the calling program, restore them before returning to the calling program, and identify itself to the calling program.

Name	Operation	Operand
	SAVE	$(r_1 [,r_2]),[T] \left[,\left\{\begin{matrix}\text{entry} \\ * \end{matrix}\right\}\right]$

r_1 ,r_2

Specifies the range of general registers whose contents are to be stored in a save area. The save area has been defined for this, the called program, by the calling program. The address of the save area has been loaded into General Register 13 by the calling program. The r_n are decimal integers in the range 0-15 (but not 13). This operand can be specified as one integer or as two integers separated by a comma. The range is usually stated as from 14 to 12, that is 14, 15, and 0 through 12 (as used in the STM Assembler language instruction).

When a register is in the range specified by this operand, its contents are always saved in a particular word in the save area. The contents of General Register 14 and 15 (if in the range specified) are saved in words 4 and 5 of the save area. The contents of General Registers 0 through 12 (if in the range specified) are saved in words 6 through 18 of the saved area. When a register is not in the range specified by this operand, its save word in the save area remains unmolested.

The range of registers specified must not contain Register 13. If the called routine is to use register 13, it must save its contents and restore the contents before issuing the RETRN macro (or terminating the program). (SAVE cannot be used to save the contents of General Register 13.)

T

Specifies that the contents of registers 14 and 15 are to be saved in words 4 and 5 of the save area, if not already saved by the r_1 and r_2 operands. If the T and r_2 operands are present and r_1 operand is 14, 15, 0, 1, or 2, the contents of all registers from 14 through the r_2 value are saved.

entry

Specifies the identifier of the entry point at which the SAVE macro is loaded. Entry can contain up to 155 characters; commas and blanks are not allowed. This operand can be a complex name, for example a combination of a file and program name.

The Assembler places this entry point identifier in the macro expansion, starting at a halfword preceding the entry point, and positioned so that one or two bytes separate its end from the beginning of the entry point. If the separation is two bytes, the Assembler places a blank character $(40)_{16}$ in the leftmost byte. The assembler places the byte count (the size) of the entry point identifier in the byte immediately preceding the entry point. This count includes the size of the identifier plus one byte for the blank character (if one has been inserted) plus one byte for the byte containing the byte count.

Written in place of an identifier when the name field of this macro instruction is to be used as the entry point identifier. If the name field is blank then the name of the control section containing the SAVE macro instruction is used.

Programming Notes

The expansion of the SAVE macro does not contain a Supervisor Call (SVC) instruction. All of the functions of SAVE can be performed by the user by programming techniques available to him in the Assembly language. However, the SAVE and RETRN macros provides a standardized method of establishing linkage into and out of a subroutine.

Examples

Example 1

Name	Operation	Operand	
EX1	SAVE	(2,10),T,F4RTNA7B99	
	DC	OH	Macro Expansion
	DC	CL11 'F4RTNA7B99', FL1'12'	
EX2	STM	14,10,12(13)	

Here, EX1 saves the contents of registers 14 through 10. The contents of registers 14, 15, 0, and 1 are saved because the T operand was written. The entry point identifier is F4RTNA7B99.

Example 2

Name	Operation	Operand	
EX3	SAVE	(3,4),,*	
	DC	OH	Macro Expansion
	DC	CL3'EX3',FL1'4'	
EX4	STM	3,4,32(13)	

In this example, EX3 saves register 3 and 4. The entry point identifer is EX3.

SETBF Macro

The SETBF (Set Buffer) macro instruction is issued in a user's program to change the size of the terminal buffer. The buffer size may range from 80 to 1024 bytes.

Name	Operation	Operand
	SETBF	{size[,N] / (1)}

size

An integer between 80 and 1024 which defines the size of the terminal buffer desired.

N

Indicates no change is to be made in the buffer size if the requested size (size defined above) is at least as large as the current buffer size.

(1)

Indicates the user has loaded General Register 1 with the size of the buffer desired before issuing the SETBF macro.

Programming Notes

If the program using the SETBF macro is in the nonconversational mode, no action will be taken because the SYSIN and SYSOUT files will handle a record up to 1020 bytes.

Cautions and Errors

General Register 15 Return Codes

x'00'	Requested buffer obtained.
x'04'	Invalid buffer size provided.
x'08'	New buffer could not be obtained.

SETIC Macro

SETIC (Set Time Block) is used in conjunction with the STXIT macro to simulate an interval timer interrupt.

Name	Operation	Operand
	SETIC	time

time

The symbolic name of the leftmost byte of the six-byte area in the user's program that contains the time interval.

The time interval must be in zoned decimal format as hhmmss, where:

hh = hours (two bytes)
mm = minutes (two bytes), and
ss = seconds (two bytes).

When the macro is issued, the Executive sets the timer to the specified interval. Decrementing of this value begins immediately. When the interval expires (clocks down to zero) control is given to the user's interval timer contingency routine specified in the STXIT macro and the Executive resets the timer to this interval again.

The user can change the time interval by issuing this macro again with a new value specified. If the user specifies a zero or negative value, the timer is turned off.

The first (leftmost) digit of the value specified for the interval is ignored. For example, the maximum allowable interval is n9 99 99, which equals 10 hours, 40 minutes and 39 seconds.

The clock used for measuring the time interval is incremented only when the task loses control of the CPU. The difference between the given interval and the effective interval can be plus 1/2 of a time slice at most.

Programming Notes

If the user program has not specified a timer routine when issuing the STXIT macro, the program is terminated when the elapsed time clock interrupt occurs.

Example

```
Name      Operation   Operand
          SETIC       INTRVL
          .
          .
          .
INTRVL    DC          X'F0F1F0F0F0F0'
```

Note: The time interval, one hour, is set in the timer to cause an elapsed time clock interrupt every hour.

SETSW Command

SETSW provides the user with the ability to set, reset, and invert the 32 task switches that he has at his disposal. SETSW can be issued either in the conversational or nonconversational mode. The 32 switches are 0-31.

Name	Operation	Operand
	SETSW	[ON=(n1,n2, ...)] [,OFF=(n1,n3, ...] [,INVERT=(n1,n2, ...)]

ON=n1, n2, ...

Indicates that the named switches are to be set to a one bit ("on" condition).

OFF=n1, n2, ...

Indicates that the named switches are to be set to a zero bit ("off" condition).

INVERT=n1, n2, ...

Indicates that the named switches are to be set on if off or off if on.

n1, n2, ...

The numbers of the task switches to be set, reset, or inverted. There are 32 task switches numbered decimally from 0 to 31.

Programming Notes

The system sets all 32 switches off when it initiates a task. The system sets switches 16-31 off when it encounters a STEP command.

If no operands are given, those task switches which are on will be noted on SYSOUT.

Examples

```
/.NAME SETSW ON=(1,3,31)  ①

/      SETSW OFF=(2,4,30)  ②

/SETSW INVERT=(7,9,11)  ③

/SETSW OFF=(1), INVERT=(2),ON=(0)  ④
```

① In this named command, the user sets his task switches 1, 3, and 31 on.

② The user sets his task switches 2, 4, and 30 off.

③ The user inverts the settings of his task switches 7, 9, and 11. If any switch is on, it is turned off. If off, it is turned on.

④ The user sets his task switch 1 off, switch 0 on, and switch 2 to the opposite of its setting.

SETSW Macro

SETSW sets, resets, and inverts task switches.

The task switches are 32 bits maintained for each task by the Executive. The 32 bits are initially set to 0. The user can set, reset, or invert any named task switches or specify the setting for all 32 task switches.

Name	Operation	Operand
	SETSW	[{ON / OFF / INVERT} =(n)$_1$, ...) / blank]

ON, OFF, INVERT

Indicate that the named switches are all to be set (on, 1) reset (off, 0), or inverted (reversed).

n, ...

The number of the task switches to be set, reset, or inverted. The n_k are decimal integers in the range 0-31. Any number (up to 32) of the task switches may be thus specified, separated by commas and enclosed in parentheses. The switches can be specified in any order. Two ascending numbers separated by a hyphen will indicate the range of switches to be set, e.g., 5-8.

blank

If no operand is specified then the user has loaded General Register 0 with the new settings that he desires. The system stores the contents of General Register 0 as the new settings of all 32 task switches.

Programming Notes

1. All task switches are automatically set to 0 (off) when a task is initiated.

2. Switches 16 through 31 are set to 0 (off) each time a STEP command is processed.

3. The GETSW macro is used to store the task switches into General Register 0.

Examples

Example 1

Name	Operation	Operand
	SETSW	ON=(17,19) ①
	SETSW	INVERT-(7,11) ②
	SETSW	OFF=(5,31) ③

Notes:

① Task switches 17 and 19 are set on (1).

② Task switches 7 and 11 are inverted.

③ Task switches 5 and 31 are set off (0).

Example 2

Name	Operation	Operand
	SETSW	

Note: The user has loaded General Register 0 with the desired switch settings for bits 0-31.

SKIP Command

This command tests the task's 32 (0-31) task switches. If conditions are met, the next command executed is that specified in the operand. The SKIP direction is forward only. This command is used primarily in the nonconversational mode, but if entered from a terminal, the conditions are noted (that is, whether met or not met) but no shipping takes place.

Name	Operation	Operand
	SKIP	.name [,ON=n1,n2, ...)] [,OFF=(n1,n2, ...)]

.name

Specifies the name of the command to be executed next if the indicated switch settings are met as stated in the command. This is a required entry. However, if the SKIP command is entered from a terminal, no skipping is done.

ON=n1,n2, ...

Indicates that the switches indicated by number are to be tested for a one bit ("on" condition).

OFF=n1,n2, ...

Indicates that the switches indicated by number are to be tested for a zero bit ("off" condition).

n1,n2, ...

The numbers of the switches to be tested. These are decimal integers in the range 0-31.

Programming Notes

When the system receives the SKIP command, it types the setting of the switches that the user has requested to be tested. If the user has entered the SKIP command from any device type other than a terminal and if the switches are set and/or reset as stated in the command, the system passes over all commands in the command input stream until the command named in the SKIP is reached. The system then accepts this named command as its next command.

If the user enters the SKIP command from a terminal, the system does not "skip." It accepts the next command in the command input stream after having typed the settings of the requested switches.

Examples

```
/.NAME SKIP .NEXSTEP,ON=(0),OFF=(1)  ①

/SKIP .NAME, OFF=(2,1),ON=(3,15,17)  ②
```

① This named command causes the system to bypass all following commands until the command whose name field is .NEXSTEP is encountered, if task switch 0 is on and task switch 1 is off. If the switches are not set as stated, the command following this one is executed.

② The system tests this task's switches. If No. 1 and No. 2 are off and if No. 3, No. 15, and No. 17 are on, it finds the command whose name field is .NAME. This command is then executed next. If the switches are not so set, the system executes the next command following the SKIP command. If the system skips and fails to find a command named .NAME, it will find the LOGOFF command and terminate the task.

SPEXT Macro

SPEXT (Supply Executive Table) enables a Class II program to obtain a Status Block which contains information required by the user's interrupt routines. SPEXT is used within a user's contingency routine. Class I programs have the equivalent of the SPEXT macro in the ADEXT macro, which supplies the address of the TOS Executive Communication Area and the address of the program table entry.

Name	Operation	Operand
	SPEXT	{address / (1)}

address

Specifies the symbolic address of a 13-byte area in the user's program into which the system is to place the Status Block.

(1)

Indicates that General Register 1 has been loaded with the address of the area into which the system is to place the Status Block.

Programming Notes

1. The contents of the Status Block are described in the following table.

Byte	Contents
0-3	The program counter (P1 setting at the time of interrupt.
4-7	The contents of General Register 10 at the time of interrupt.
8-11	The contents of General Register 11 the time of interrupt.
12	The weight code associated with the specific interrupt.

2. SPEXT is ignored in a Class I program.

STATUS enables the user to determine the status of tasks being processed. The facility is provided to obtain the status of one or all user tasks. The status report is typed on the user's terminal, or, if entered by the operator, his console typewriter.

Eight kinds of status reports can be requested, using STATUS. Only one kind can be requested at a time; therefore, only one of the three operands (SUMMARY, LIST, TSN) can be used in one STATUS command.

Name	Operation	Operand
	STATUS	SUMMARY[,userid] LIST[,userid] [,TYPE=] PROG[,userid] [,TYPE=n] TSN[,ITN] [,TIME] [,PROG] SATQ DVQ BIAS ITN MSG RJOB no operand

SUMMARY

Provides a summary of the number of each of the following eight types of tasks:

Type 1: tasks in the job pool.

Type 2: nonconversational tasks running.

Type 3: conversational tasks running.

Type 4: tasks queued for spooling out.

Type 5: tasks being spooled out.

Type 6: remote batch tasks being spooled out.

Type 7: DXC tasks being spooled out.

Type 8: remote batch terminals.

userid

Used when the summary information is requested for a particular user. Userid consists of from one to eight alphanumeric characters. The first character must be alphabetic.

LIST

Provides the following information for all tasks being processed:

1. task sequence number

2. priority

3. type (1-5, as defined under SUMMARY above)

4. terminal (buffer) number

5. CPT time used for all tasks being processed.

LIST cannot be used if SUMMARY or TSN is used.

userid

Used when the list information is requested for a particular user. Userid consists of from 1 to 8 alphanumeric characters.

TYPE=

Used to restrict the LIST information to one task type instead of all tasks being processed.

n

This is one digit ranging from 1 to 5, as defined SUMMARY.

TSN=

Provides the following information for the task whose task sequence number is specified:

1. time that the task was spooled in.

2. time that the LOGON command was processed.

3. time that the LOGOFF command was processed.

4. time that the task was spooled out.

5. CPU time used by the task.

6. task priority

7. task type (as defined under SUMMARY above).

tsn

Task sequence number for which the information listed under TSN is requested; tsn consists of four decimal digits in the range 0000-9999.

SATQ

Provides a list of TSN's on the five system saturation queues — core overflow, core hold, drum overflow for background tasks, drum overflow for interactive tasks, and drum hold queue.

DVQ

Lists background TSN's awaiting completion of SECURE command device reservation and the actual devices required for each TSN.

BIAS

Provides current core bias allowed, current number of active terminals allowed, current number of background tasks allowed, current drum bias allowed, and current maximum time each task is allowed to run.

ITN

Allows the system controller to obtain the TSN of a task using only its internal task number.

MSG

Lists TSN's having console messages pending. The message will be listed with each TSN. This is only an operator command.

RJOB

Provides the console operator with TSN, userid, station-id, line number, and disposition of remote batch tasks.

no operand

Provides the user with his TSN, current time, logon time, and buffer number.

Programming Note

This command can only be issued in the conversational mode.

Examples

1. /STATUS SUMMARY

```
%0.165546     TYPE1  TYPE2  TYPE3  TYPE4  TYPE5  TYPE6  TYPE7  TYPE8
%0.165546     0000   0000   0002   0000   0000   0000   0000   0000
```

2. /STATUS LIST

```
%0.161556     TSN    PRI  TYPE  TERM  CPU-TIME    USERID
%0.161556     0171    09   2           0000.0901   TSOS
%0.161556     0172    09   2           0000.1253   TSOS
%0.161556     0173    09   2           0000.1662   TSOS
%0.161603     0174    09   2           0000.1764   TSOS
%0.161608     0175    09   2           0000.1301   TSOS
```

3. /STATUS PROG

```
%0.161800     TYPE  PRI  CURR-CMD  PROG    SIZE  CLASS
%0.161800      2     09            EDT     001     2
%0.161800      2     09            BASIC   002     2
%0.161800      2     09            BASIC   001     2
%0.161807      2     09  FILE      EDT     003     2
%0.161813      2     09            ASMDOS  001     2
%0.161820      2     09            ASMDOS  002     2
```

4. /STATUS BIAS

```
%0.161857     CORE  TERM   NC  DRUM  TIME
%0.161857     0024  0010  0010 0062  655.00
```

5. /STATUS DVQ

```
%0.161137  0172    T9 N= 03,WORK=05,L1,T0,T1,T2,
%0.161137  0174    T9 N= 03,WORK=05,L1,T0,T1,T2,
%0.161137  0175    T9 N= 01,T7=  01,T7DC=01,D564=01,WORK=02,D590=01,
T9P= 01,L1,T0
%0.161137  0176    T9 N= 01,T7=  01,T7DC=01,D564=01,WORK=02,D590=01,
T9P= 01,L1,T0
```

6. /STATUS 234,PROG

```
%   TYPE  PRI  CURR-CMD  PROG   SIZE  CLASS
%    3     09            EDT     002    2
```

7. /STATUS 234,ITN

```
%   ITN  VIR-ADDR PEND   QUE#
%    10   1CADE0   2       12
```

8. /STATUS 234,TIME

```
%   TYPE  SPOOLIN  LOGON  LOGOFF  CPU-TIME
%    3     1101     1103    0000   0010,3444
```

```
 9.  /STATUS SATQ

     %0.165645 SATQ1=0
     %0.165645 SATQ2=0
     %0.165645 SATQ3=0
     %0.165645 SATQ4=0
     %0.165648 SATQ5=0

10.  /STATUS MSG

     %?234 THIS IS A TEST

     /STATUS RJOB

     %   TSN  STAT-ID  USER-ID  LINE  DISPOS
     %   234   A1078    TSOS     25   RCA 740

11.  /STATUS ITN=X'0A'

     %   234

12.  /STATUS

     %   TSN  CPU-TIME   LOGON  BUFF
     %   234   0006.1523  1543   37
```

STEP Command

STEP defines a logical subdivision of a task. When a program has terminated abnormally, the system automatically SKIPs (see SKIP Command) to the next STEP command in the command input stream (or, if there is no STEP command following, to the LOGOFF command). STEP is used for nonconversational tasks or for conversational tasks in the PROC mode and resets the Monitor Table (refer to "VMOS Support of TOS Tables" in the appendices) and task switches 16-31.

Name	Operation	Operand
	STEP	

Programming Notes

The system resets task switches 16-31 to zero when it encounters a STEP command.

Examples

```
/.NAME STEP  (1)

/STEP  (2)
```

(1) This STEP command has a name field, for possible use with the SKIP command.

(2) The Monitor Table is reset and task switches 16-31 are set off.

STXIT Macro

The STXIT macro (Set Contingency Address) instruction enables a Class I or Class II program to specify addresses of from one to four interrupt routines to which the VMOS Executive gives control when any of the specified interrupts occur.

The four interrupts are:

1. program check,

2. interval timer,

3. operator communication, and

4. unrecoverable program error.

The STXIT should be one of the first instructions executed within the program. When the STXIT is executed, the VMOS Executive stores the address of any specified routines. When one of the given conditions occurs, the Executive transfers control to the associated contingency routine.

If the user does not specify one of the routines, the address stored by a previous STXIT is assumed. If a previous STXIT was not issued, the ABORT or CLOSE operand (as explained below) is assumed.

Name	Operation	Operand
	STXIT	[R],[{prochk / CLOSE / ABORT}],[{timer / CLOSE}], [{oprtn / CLOSE}],[{error / CLOSE}],[{runout / CLOSE}]

R

This operand is included for source language compatibility. This entry is ignored by VMOS, but note that the comma following it is required.

prochk

Specifies the symbolic address in the user's program of the routine to handle a program check interrupt. The following conditions cause a program check interrupt:

1. Operation Code Trap

2. Illegal SVC (or parameter error)

3. Exponent Overflow

4. Device Error

5. Significance Error

6. Exponent Underflow

7. Decimal Overflow

8. Fixed Point Overflow

9. Data Error

timer

Specifies the symbolic address in the user's program of the routine to handle the interval timer interrupt. The interval timer can be set with the SETIC macro.

error

Specifies the symbolic address in the user's program of the routine to handle an unrecoverable program error interrupt. The following conditions cause an unrecoverable program error interrupt:

1. privileged operation

2. address error

3. paging error

ABORT

Applicable only to the program check interrupt, ABORT has the same meaning as CLOSE.

CLOSE

Causes the program to be aborted if the specified error occurs, except for an operator communication interrupt which is ignored. The next job is then initiated.

error

Specifies the address of the user's operator interrupt routine or the word CLOSE.

runout

Specifies the address of the user's program-time-runout routine. Nonconversational tasks are guaranteed a 30-second cleanup period.

Programming Notes

Program Check Interrupt

When a program check interrupt occurs, the Executive does one of the following:

1. If the program has not specified an address of a program check routine or if the keywords CLOSE or ABORT were specified, a message is typed to the operator indicating the error, and the job is terminated. The operator is given the option of dumping the program before it is terminated.

2. If the program has specified an address of a program check routine:

a. The Executive stores the P1 counter in the program's Executive Storage Area.

b. The Executive stores the contents of General Registers 10 and 11 in the program's Executive Storage Area. These registers are available for the program check routine to use.

c. The interrupt weight code (indicating the specific interrupt that occurred) is stored in the program's Executive Storage Area (bytes 116-119). One of the following weight codes is stored:

Weight Code (Hexadecimal)	Meaning
00000050	Supervisor Call
00000058	Operation Code Trap
00000064	Exponent Overflow
0000006C	Significance Error
00000070	Exponent Underflow
00000074	Decimal Overflow
00000078	Fixed-Point Overflow
00000060	Data Error

d. Control is transferred to the program check routine address specified.

Note: The program check routine must terminate with an EXIT or an EXITP macro containing a PR operand entry. This causes the P1 counter and General Registers 10 and 11 to be restored to the values previously stored in the Executive Storage Area.

Interval Timer Interrupt – When an interval timer interrupt occurs, the Executive does one of the following:

1. If the program has not specified an address of an interval timer routine or the keyword CLOSE was specified, a message is typed to the operator indicating the error, and the job is terminated. The operator is given the option of dumping the program before it is terminated.

2. If the program has specified an address of an interval timer routine:

a. The Executive stores the P1-Counter in the program's Executive Storage Area.

b. The Executive stores the contents of General Registers 10 and 11 in the program's Executive Storage Area. These registers are available for the interval timer routine to use.

c. Control is transferred to the terminal routine address specified.

Note: The interval timer routine must terminate with an EXIT or an EXITP macro containing a TR operand. This causes the P1-Counter and General Registers 10 and 11 to be restored to the values previously stored in the Executive Storage Area.

Operator Communication Interrupt – When an operator communication interrupt occurs, (the user executes the INTR command), the Executive does one of the following:

1. If the program has not specified an address of an operator-communication routine or the keyword CLOSE was specified, the interrupt is ignored.

2. If the program has specified an address of an operator communication routine:

a. The Executive stores the P1-Counter in the program's Executive Storage Area.

b. The Executive stores the contents of General Registers 10 and 11 in the program's Executive Storage Area. These registers are available for the operator-communication routine.

c. Control is transferred to the operator communication routine address as specified.

Notes:

1. The TYPE or the TYPIO macro can be used to communicate between the program and the operator.

2. The operator-communication routine must terminate with an EXIT or an EXITP macro containing a CR operand. This causes the P1-Counter and General Registers 10 and 11 to be restored to the values previously stored in the Executive Storage Area.

3. If the operator attempts to interrupt a program that is already servicing an operator communication (an EXIT-CR macro has not been issued), the interrupt is ignored.

Unrecoverable Error Interrupt – The following conditions cause an unrecoverable error interrupt to occur:

Privileged Operation

Address Error

Paging Error

When an unrecoverable error interrupt occurs, the Executive does one of the following:

1. If the program has not specified an address of an unrecoverable error interrupt routine or the keyword CLOSE was specified, a message is typed to the operator indicating the error, and the job is terminated. The operator has the option of dumping the program before it is terminated.

2. If the program has specified the address of an unrecoverable error routine:

   a. The Executive stores the P1-Counter in the program's Executive Storage Area.

   b. The Executive stores the contents of General Registers 10 and 11 in the program's Executive Storage Area. These registers are available for the unrecoverable error routine to use.

   c. The interrupt weight code (indicating the specific interrupt that occurred) is stored in the program's Executive Storage Area (bytes 116-119). One of the following weight codes is stored:

Weight Code (Hexadecimal)	Meaning
00000054	Privileged Operation
0000005C	Address Error

   d. Control is transferred to the unrecoverable error routine address specified.

   Note: The unrecoverable error routine must end with an EXITP macro (or a TERM or TERMD macro) containing a UR operand.

Example

```
Name        Operation    Operand
            STXIT        ,PCERR,CLOSE,,UNREC
```

The STXIT in this example indicates that,

1. If a program check error occurs, control is to be given at a routine PCERR,

2. If a timer interrupt occurs, the program is to be aborted, and

3. If an unrecoverable error occurs, control is to be given to the routine, UNREC.

TERM Macro

TERM macro is issued to terminate a program. All input/output operations initiated by the program before the execution of the macro are completed prior to program termination.

Name	Operation	Operand
	TERM	

When this macro is executed, the following occurs:

1. All devices assigned to the program are deallocated.

2. Memory assigned to the program is deallocated.

3. The TOS Operation List and TOS Program Table entries are marked unassigned.

4. The operator is notified of the termination.

Device List – Bytes 8-30 for each deallocated device are cleared.

Operation List – The first byte is made $(FF)_{16}$.

Program Table Entry – Program base address (four bytes) is set to zeros.

TERMD Macro

TERMD macro is issued to terminate a program and obtain a program dump at termination. All input/output operations initiated by the program prior to the execution of the macro are completed.

Name	Operation	Operand
	TERMD	

When this macro is executed, the following occurs:

1. Scratch-Pad memory, the TOS Executive, the program's TOS Program Table and Device List entries, and the program including its Executive Storage Area and Run Time Parameter Area (if any) are dumped to the SYSLST file.

2. The program is terminated as follows:

   a. All devices assigned to the program are deallocated.

   b. Memory assigned to the program is deallocated.

   c. The Operation List and Program Table entries are marked unassigned.

   d. The operator is notified of the termination.

Device List – Bytes 8-30 for each deallocated device are cleared.

Operation List – The first byte is made $(FF)_{16}$.

Program Table Entry – Program base address (four bytes) is set to zeros.

TERMJ Macro

TERMJ macro is issued to terminate a job within a task.

Name	Operation	Operand
	TERMJ	

Programming Notes

If TERMJ is issued by a program running in a nonconversational task, control is passed to the first STEP or LOGOFF command encountered in the command input stream. If a LOGOFF command is encountered, control is passed to the job scheduler for the selection of the next job to be processed.

If the program was running in a conversational task when the TERMJ macro was issued, the next command is read.

TMODE Macro

TMODE (Task Mode) provides the user with information about his task. The specified data provided are named and described below. The user specifies a location in his program where the information is to be stored.

Name	Operation	Operand
	TMODE	area[,length]

area

Specifies the symbolic address of the area in the user's program in which the system is to store the task information.

length

Specifies the size of the area in which the task information is to be stored. If the length operand is omitted, the length attribute of the area is used. If, in either case, the length is less than the number of bytes of information described below, the data is truncated by the difference, the last named data being dropped first.

The following information is stored in the user's area:

Byte	Symbolic Name	Description
0	TSKTYPE	Task type, and if conversational, terminal type. If the byte is zero, the task is nonconversational. Otherwise, the value in the byte corresponds to a terminal device type, as follows: X'01' – Teletype model 33 X'02' – Teletype model 35 X'03' – Teletype model 37

Byte	Symbolic Name	Description
		X'04' – 8752 Video Data Terminal
		X'05' – IBM 2741 Communications Terminal
		X'10' – DCT 2000 Data Communications Terminal
		X'11' – 8740-41 Remote Batch Terminal
1-2	TSKBUFSZ	Task buffer size for conversational tasks. The size of the input buffer which has been allocated at LOGON, or changed by SETBF.
		(TSKBUFSZ is not applicable to nonconversational tasks.) TSKBUFSZ is specified during LOGON and is stored as a binary number.
3	TSKPRI	Task priority. TSKPRI is stored as a binary number.
4-7	TSKTSN	Task sequence number. TSKTSN is stored as a zoned decimal number.
8-15	TSKUSER	The user identification code as supplied at LOGON.
16-23	TSKACCT	The 8-byte account number to which this task is charged. The account number is in EBCDIC, left justified, and space filled when necessary.
24-27	TSKTIM	The amount of CPU time used by this task up to the present time. TSKTIM is stored as a binary number, in units of hundreds of microseconds.
28	TSKPRV	The privilege code of the task. The codes are:
		$(01)_{16}$ – System Administrator. $(02)_{16}$ – User.

Example

Name	Operation	Operand
	TMODE	TASKINFO
	.	
	.	
	.	
TASKINFO	DS	CL24

Note: The length attribute of TASKINFO, 24_{10}, is used to determine the amount of data to be stored at location TASKINFO. Bytes 0-23 are stored.

TOCOM Macro

TOCOM moves data to the Common Data Area from the user program. This area is maintained by the Executive and is common to all Class I and Class II programs. Data moved into the Common Area remains in this area until system shutdown or until replaced by data moved in by another issuance of the TOCOM Macro.

Name	Operation	Operand
	TOCOM	location,data[,length]

location

A decimal number indicating the location relative to the first (designated by 1) byte of the Common Data Area into which the first byte of data is to be moved.

data

The symbolic address in the user program of the first byte to be moved to the Common Data Area.

length

The number (as a decimal) of bytes to be moved. If length is omitted the implied length of the data operand is used.

If an addressing error occurs because the user did not specify at SYSGEN time that a Common Data Area was to be used or because the location plus the length attribute exceeds the bounds of the Common Data Area, return is made to the program check routine (if one was specified by issuance of a STXIT macro) indicating an illegal SVC. If no STXIT has been issued, and an addressing error occurs, the program is terminated with an illegal SVC indication.

VPASS Macro

VPASS (Variable Length Pass) is used to relinquish control of the processor for one or more seconds.

Name	Operation	Operand
	VPASS	{seconds / (1)}

seconds

Specifies the number of seconds to relinquish control of the processor. The value must lie in the range $1 \leqslant$ seconds $\leqslant$ 21599.

Programming Notes

If the value specified in Register 1 is less than 1, a 1 second VPASS will be executed; if greater than 21599 (5 hours, 59 minutes, 59 seconds), 21599 is used.

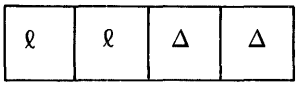
WRLST Macro

WRLST macro (Write Print Line) causes a record to be written to SYSLST. The Executive creates a temporary SYSLST EAM file upon receiving the first WRLST instruction from the user, unless a PROUT macro was previously issued. Thereafter, each WRLST instruction in the user's program causes one record to be written to the SYSLST file. The user specifies the address of the record and the address of an error routine to which the Executive transfers control when an error occurs during execution of the WRLST macro.

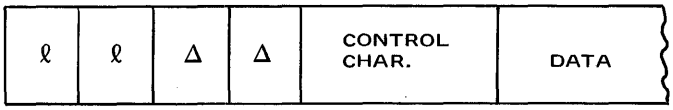
Name	Operation	Operand
	WRLST	{record,error / (1)}

record

Specifies the symbolic address of the V-type record to be written to SYSLST. A V-type record has as its first four bytes a count field that contains the size in bytes of the record plus 4, i.e., the count field appears as



The first character of the record proper must contain a print control character. The print control character is one byte of control information that specifies the type of paper advance desired when the SYSLST file is spooled out to the printer. The entire record looks like



The valid print control bytes are listed below:

$(4n)_{16}$

Space immediately n lines. The range of n is 0-F representing 0-15 lines skipped. Setting n=0 causes space suppression.

$(0n)_{16}$

Space n lines after the next print. The rules for n are as described below.

$(Cn)_{16}$

Skip immediately to channel n on the carriage control loop. The range of n is 1-B, excluding 9.

$(8n)_{16}$

Skip to channel n on the carriage control loop after the next print. The range of n is 1-B excluding 9.

error

Specifies the symbolic address of the routine in the user's program to which the Executive gives control when one of the following errors occurs during execution of the WRLST macro:

nonrecoverable error

parameter error

truncation error (the record is greater than 1020 bytes.)

(The Executive loads General Register 15 with a return code before transferring control to the error routine.)

(1)

Indicates that the user has loaded General Register 1 with the address of a parameter table which contains the addresses of the record to be written to SYSLST and the error routine.

The contents of this parameter table are as follows:

Byte	Parameter
0	Reserved
1-3	Address of the V-type record to be written to SYSLST.
4	Reserved
5-7	Address of the error routine in the user's program.

Cautions and Errors

General Register 15 Return Codes:

X'04'	nonrecoverable error
X'08'	parameter error
X'0C'	write truncation (the V-type record less five bytes, four for the count field, one for the print control character, is too large for the device buffer.)

Programming Notes

1. On normal termination, control is returned to the user at the location following the macro. When a nonrecoverable error, parameter error, or truncation occurs, control is passed to the error address with the return code loaded into register 15. Since Spoolout (see VMOS Service Routines Manual) uses the default condition of 132 characters in spooling out the SYSLST file, truncation occurs when the length of the record exceeds 137 bytes (4-byte length field + 1-byte print control + 132 characters) and no record is written. Register 14 will contain the address which follows the macro, so error processing routines can return to in-line processing. Register 15 is unchanged when no error occurs.

2. SYSLST is spooled out onto a line printer at task termination (except when the output is destined for a remote station). After spooling, the SYSLST file is erased.

WROUT Macro

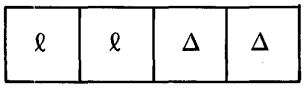
WROUT (Write Records to SYSOUT) sends a message from a program to the SYSOUT EAM file. SYSOUT may be a terminal or a system work file. The user specifies the address of the message to be written to SYSOUT, the address of the error routine to which the Executive transfers control when an error occurs during execution of the WROUT macro, and optionally, a choice of edit controls for the terminal. The user can formulate his own parameter area and load its address into General Register 1. In this case, he then uses the second format of the macro instruction.

Name	Operation	Operand
	WROUT	{ message,error[,edit] } { (1) }

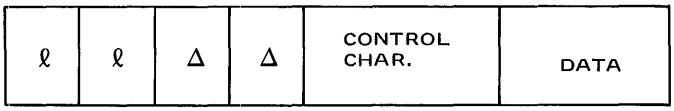
message

Specifies the symbolic address in the user's program of the message to be written to SYSOUT.

The message must be a V-type record. A count field containing the size in bytes of the physical record plus 4, must precede the first four bytes of the record. The count field appears as



The first character of the record proper must contain a print control character. The print control character is one byte of control information that specifies the type of paper advance desired when the SYSLST file is spooled out to the printer. The entire record looks like



The valid print control bytes are listed below:

$(4n)_{16}$

Space immediately n lines. The range of n is 0-F representing 0-15 lines skipped. Setting n=0 causes space suppression.

$(0n)_{16}$

Space n lines after the next print. The rules for n are as described above.

$(Cn)_{16}$

Skip immediately to channel n on the carriage control loop. The range of n is 1-B, excluding 9.

error

Specifies the symbolic address of the routine in the user's program to which the Executive gives control when one of the following errors occurs during execution of the WROUT macro:

1. nonrecoverable error
2. parameter error
3. write truncation
4. break

The Executive loads an error return code into General Register 15 before returning control to the error routine. General Register 14 contains the address of the first byte which follows the macro expansion.

edit

Specifies the editing option for the record to be written to the terminal. The options are:

$(00)_{16}$

Translation from EBCDIC to ASCII and insertion of line feed and carriage return characters into the output text.

$(01)_{16}$

Insertion of line feed and carriage return characters into the output text.

$(02)_{16}$

Translation from EBCDIC to ASCII.

$(03)_{16}$

No editing is done.

The edit option is ignored if SYSOUT is not a terminal.

(1)

Indicates that the user has loaded General Register 1 with the address of a parameter table which contains the address of the message to SYSOUT, the address of the error routine, and the edit option code.

The format of this parameter table is as follows:

Byte	Parameter
0	Edit option code.
1-3	Address of the V-type record to be written to SYSOUT.
4	Reserved.
5-7	Address of the error routine in the user's program.

Programming Notes

When issued during a DXC session, the WROUT macro will cause an MT=7 Message Type to be transmitted to the auxiliary processor. MT=7 causes the auxiliary processor to write the record as it is received.

Cautions and Errors

General Register 15 Return Codes:

X'04'	Nonrecoverable error.
X'08'	Parameter error.
X'0C'	Write truncation (the length of the V-type record, less five bytes, four for the count field, one for the print control character, is greater than the terminal buffer size.) The excess characters are not written. The other characters are written.
X'10'	Break. The user has pressed the break key during execution of the WROUT macro.

WRTOT Macro

WRTOT (a TOS Monitor Macro) is used to write system output (a record to SYSOPT). If the SYSOPT device has not been allocated, the first WRTOT received by the VMOS Executive causes the allocation of a temporary SYSOPT work file for this task. SYSOPT is spooled out at task termination.

The system converts the user's output records to V-type records by prefixing a 4-byte count field to the record before writing them to SYSOPT.

Name	Operation	Operand
	WRTOT	address[,length]

address

Specifies the symbolic address in the user's program of the record to be written to SYSOPT.

length

Specifies the decimal number of characters to be written. The maximum length is 80 bytes. If the length operand is omitted, the length attribute of the area is taken as the length.

Cautions and Errors

If a nonrecoverable error occurs during execution of the WRTOT macro, the program is terminated and control passes to the next STEP or LOGOFF command.

WRTRD Macro

The WRTRD macro (Terminal Tandem Write/Read) instruction is used by programs in the conversational mode to:

1. Send a message to the terminal, and

2. Accept a response from the terminal.

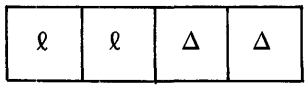
A typical use for the WRTRD macro is to prompt the conversational user and receive a response from him.

The user can formulate his own parameter area and load its address into General Register 1. In this case, he uses the second format of the macro instruction.

Name	Operation	Operand
	WRTRD	message1,[edit1],message2 ,[edit2],[length],error
	WRTRD	(1)

message1

Specifies the symbolic address in the user's program of the message to be written to the terminal. The message must be a V-type record (the first four bytes contain the length of the record). The next byte following the 4-byte count field is reserved. The message proper follows these five bytes. The count field appears as



edit1

This operand specifies the editing option for the record to be written. Refer to "Macro Instruction Editing Options" in the appendices for further information.

If this operand is omitted, 03 is assumed.

message2

Specifies the symbolic address in the user's program into which the message from the terminal is to be placed. The Executive puts the input message into the message2 area as a standard V-type record (for example, first four bytes contain a count of the number of bytes in the entire record followed by the data). This area may be the same as the output message area, message1.

edit2

Specifies the editing option for the record to be read. Refer to "Macro Instruction Editing Options" in the appendices for further information.

length

Specifies the size of the input area to receive the message from the terminal. This size must include four bytes for the V-type record length field. If this operand is omitted, the system assumes the length attribute of the area to be the length of the message.

error

This operand specifies the symbolic address of the routine in the user's program to which the Executive gives control when one of the following errors occur during execution of the WRTRD macro:

1. nonrecoverable error

2. parameter error

3. read truncation

4. write truncation

5. nonconversational mode

6. end of file

Before returning control to the user, the Executive loads General Register 15 with an error return code. General Register 14 contains the address of the first byte following the macro expansion.

(1)

This operand indicates that the user has loaded General Register 1 with the address of a parameter table which contains the following information:

Byte	Parameter
0	Edit option of the message to be written.
1-3	Address in the user program of the message to be written.
4	Edit option of the response message from the terminal.

Byte	Parameter
5-7	Address in the user program of the response message from the terminal.
8-9	Reserved.
10-11	Length of the message to be read.
12	Reserved.
13-15	Address in the user program of the error routines.

Programming Notes

When a break occurs during the write/read operation, automatic rollback occurs. Automatic rollback is the resetting of the user's P-counter back to the macro, so that after the break interrupt is processed the macro is repeated.

When issued during a DXC session, the WRTRD macro will cause an MT=8 Message Type to be transmitted to the auxiliary processor. MT=8 causes the auxiliary processor to write the record as it is received and be prepared to read a record from the terminal.

Cautions and Errors

1. General Register 15 Return Codes:

X'04' nonrecoverable error.

X'08' parameter error.

X'0C' read truncation. The size of the record exceeds the designated length (after allowing four bytes for the length field). The last part of the record is dropped.

X'10' write truncation. The length of the record written (after allowing four bytes for the length field) exceeds the terminal buffer size (normally 72 bytes). The excess characters are not printed.

X'14' nonconversational mode. The WRTRD macro has been issued in the nonconversational mode.

2. If the length of the record written (less four bytes for the length field and one byte for the reserved byte) exceeds the terminal buffer size, the record is truncated (the excess characters are not written). The user receives control at the error address with the appropriate error code in Register 15 (for example, X'10').

3. If the size of a record read exceeds the designated length (minus four bytes for the length field), the record is truncated to the size specified in the length field (less four). The user receives control at the error address with the appropriate error code in Register 15 (for example, X'10').

Example

Name	Operation	Operand
	WRTRD	MSG1,,MSG2,,60,ERR

MSG1

The symbolic tag in the user program specifying the record to be output to the terminal.

edit1

An omitted operand. $(03)_{16}$ is assumed and no editing is performed.

MSG2

The symbolic tag in the user program associated with the area into which a message from the terminal is to be placed.

edit2

An omitted operand. No editing is performed.

60

The size of the input area to receive the message from the terminal.

ERR

The tag associated with the user's error contingency routine.

GENERAL

The information contained in this section defines the explicit instructions provided by VMOS for programmer control of file processing. For convenience of use, the section has been divided into two areas: the first of which contains those commands and macros applicable to the processing of any program file, while the latter section discusses the macros specifically applicable to the various file access methods.

FILE MANAGEMENT COMMANDS AND MACROS SUMMARIES

The file management commands and macros constitute those VMOS instructional entities facilitating the creation, maintenance, and disposition of program files by the user. The following are summaries by functional category of these commands and macros. Several commands and macros will appear in more than one category. This occurs as a result of the scope of the instruction, and does not imply the existence of more than one instruction with the same name.

FILE CREATION COMMANDS AND MACROS

Command/Macro	Function
CATALOG Command CATAL Macro	Create a catalog entry.
DATA Command	Allocate, create, and catalog a file using the spoolin facilities.
FCB Macro	Define a File Control Block. See FILE command description.
FILE Command/Macro	Catalog a file and define its characteristics.
IDFCB Macro	Provide symbolic name for an FCB. See "VMOS General Service File Management Macros" in the appendices.
OPEN Macro	Establish a logical connection between a file and the problem program, complete the file control block fields, verify or create file labels, position volumes to the first record to be processed, and allocate buffer areas as required. See appendix referred to above for the discussion of the user of this macro.
PASSWORD Command	Supply a file password.

FILE MAINTENANCE COMMANDS AND MACROS

Command/Macro	Function
CATALOG Command CATAL Macro	Alter a catalog entry.
CHANGE Command CHNGE Macro	Alter or replace a file's linkname.
COPY Command/Macro	Copy a file.
DROP Command	Remove a file definition from HOLD status.
FSTATUS Command FSTAT Macro	Furnish information on the status of a file.
HOLD Command	Temporarily prevent the release of a file definition.
RELEASE Command	Delete a file definition.

FILE DISPOSITION COMMANDS AND MACROS

Command/Macro	Function
CLOSE Macro	Indicates that file processing is complete, replaces automatically-acquired buffer areas, performs label verification and creation, updates catalog entries, ensures that pending write operations are completed, and positions volumes as indicated. See "VMOS General Service File Management Macros" in the appendices for the discussion of the use of this macro.
END Command	Terminate a DATA file.
ERASE Command/Macro	Delete a file's entry from the catalog deallocate space, or make the file appear logically empty (NULL).
FILE Command/Macro	Deallocate random access space.
RELEASE Command REL Macro	Release a file definition and its associated private devices.
SYSFILE Command	Reassign SYSIPT, SYSDTA, or TASK LIB files to another file.

FILE RECONSTRUCTION COMMANDS AND MACROS

Command/Macro	Function
LOG Macro	Inform access method of intention to modify a record.
RECON Command/Macro	Request reconstruction of a specified file.
RESET Command	Request resetting of a specified file.

FILE MANAGEMENT COMMAND AND MACRO DESCRIPTIONS

The following material describes each of the file management commands and macros. The descriptions include instruction format, programming considerations, and, where illustrative information is necessary, examples of the use of the instruction. The commands and macros are presented in alphabetical order without regard to functional classification.

CATALOG Command
CATAL Macro

Both the CATALOG command or CATAL macro create or alter a catalog entry for a file. (To allocate space for a file, see File command.)

Name	Operation	Operand
	{CATALOG / CAT}	filename1-name,[filename2-name] [,STATE=NEW|UPDATE] [,ACCESS=READ|WRITE]
	CATAL	[,SHARE=YES|NO] [,RDPASS=NONE|special] [,WRPASS=NONE|special] [,RETPD=days-integer]

filename1

Specifies fully qualified name of the file to be cataloged. Only the controller may specify the $userid prefix.

filename2

Specifies the new name for the file. The userid prefix cannot be specified, since this would imply a change of file ownership. This parameter is ignored unless STATE=UPDATE.

STATE=

Specifies the function to be performed.

NEW

Specifies that a new catalog entry is to be created for a file that does not yet exist.

UPDATE

Specifies that an existing catalog entry is to be modified.

Note: There is no default case for omitted parameters in the update mode. A field in the catalog entry will not be modified unless an explicit parameter is supplied.

ACCESS=

Specifies the manner in which the file may be processed.

READ

Specifies read only access to the file.

WRITE

Specifies read and/or write access to the file.

SHARE=

Specifies whether or not the owner of the file will allow it to be accessed by other users.

YES

Specifies that the file may be shared.

NO

Specifies that the file is not sharable; thus it may be accessed only by the owner (or the controller).

RDPASS=

NONE

Specifies that no password is required to read the file.

special

Specifies the password required to read the file. The password must be a 1- or 4-byte decimal, hexadecimal, or character self-defining term. A read password of X'00000000' is ignored.

WRPASS=

NONE

Specifies that no password is required to write to the file.

special

Specifies the password required to write to the file. The password must be a 1- or 4-byte decimal, hexadecimal, or character self-defining term. A write password of X'00000000' is ignored. Note that read access will be permitted when the write password is supplied (even if the file has a read password).

RETPD=

Specifies the retention period for the file. This parameter is ignored unless STATE=UPDATE. Note that this parameter is only applicable to existing files being overwritten or updated.

Programming Notes

The retention period and filename can be changed for tape files. These changes are only made to the catalog and not to the actual labels on the tape reel.

When the tape file is opened in a mode which allows write operations, a retention period check is made against the value specified in the catalog if the file already exists. If the file does not exist (not an overwrite or update), the retention period on the reel is used. The filename, however, is checked or verified against the name contained on the tape reel. The user is cautioned that changing of the filename should follow the left parenthesis (group) convention as described in the section on file groups.

When the macro form is used, the return codes are set in register 15. 00 indicates successful completion of the operation. The codes for unsuccessful completion are specified in the IDEMS macro. Refer to "File Management Error Message Concepts and Code Structure" in the appendices.

CHANGE Command
CHNGE Macro

Both the CHANGE command or CHNGE macro change the linkname or symbolic device name of a previous file definition. No other parameters introduced by the FILE command are changed.

Name	Operation	Operand
	CHANGE	[link-symbol] { ,new link-symbol / ,symdev-symbol }

link-symbol

Specifies the file definition name.

Default: a linkname of spaces is assumed.

new link-symbol

Specifies the linkname which is to replace the existing name.

symdev-symbol

Specifies the TOS symbolic device name that is to be changed.

Programming Notes

The CHANGE command is primarily intended to allow the user to pass FILE command definitions from program to program within a task without changing the link symbols specified in the programs' FCB's.

When the macro form is used, the return codes are set in register 15. 00 indicates successful completion of the operation. The codes for unsuccessful completion are specified in the IDEMS macro. Refer to "File Management Error Message Concepts and Code Structure" in the appendices.

Either the COPY command or COPY macro may be used to copy a file.

Name	Operation	Operand
	COPY	filename1-name,filename2-name[,SAME]

filename1

Specifies the fully qualified (up to 54 bytes) filename of the file to be copied. This file is still unaltered after execution of this command.

filename2

Specifies the fully qualified (up to 54 bytes) name of the file to receive the copy.

SAME

Specifies that filename2 is to have the same file security as filename1 (for example, same passwords, sharability, retention period, and access).

Programming Notes

This command cannot be used to create a file with different file characteristics (for example, SAM to ISAM, fixed-length records to variable-length records). This fact is more evident by noting that files are copied on a half-page (2048 bytes) basis; that is, GET's and PUT's are not used to read and write logical records, respectively.

Filename2 cannot be the same as filename1. If filename2 is not cataloged, and if SAME is not specified, it is cataloged as a nonshared file with read/write access. If no device has been specified for filename2, public volumes are used. If no space has been specified for filename2, sufficient primary space to contain the file is obtained. Note that device and space assignments are obtained from the catalog entry for filename2. In this case, the value assumed for secondary space allocation is the system default.

Only the controller can specify the $userid prefix for filename2, if filename2 is not yet cataloged.

Direct access files may be freely copied to tape or to other direct access devices (for example: 8564 to 8568, 8568 to tape, 8590 to 8564). Tape to direct access devices is supported for single volume tape files. Tape-to-tape copy operations are not supported with this command. A utility program is available to provide this facility but the user must then create the appropriate catalog entry with a FILE command.

When the macro form is used, the return codes are set in register 15. 00 indicates successful completion of the operation. The codes for unsuccessful completion are specified in the IDEMS macro. Refer to "File Management Error Message Concepts and Code Structure" in the appendices.

DATA Command

The DATA command is used to allocate and catalog a file by means of the spoolin (RCARD) procedure. All cards following a DATA command will be placed in the file until an END command or subsequent DATA command is encountered, at which time the file is closed.

Name	Operation	Operand
	DATA	filename [,filetype = {SAM / ISAM}]

filename

Specifies the name by which the DATA file will be cataloged.

filetype

Specifies the access method for the file. (SAM is assumed if the operand is omitted.)

Programming Notes

The DATA command must follow the LOGON statement initially, but then it may follow an END command or data of a previous DATA command. When a subsequent DATA command is encountered, the file is closed and a new file is created. Diagnostic messages for errors will be given by the normal spoolout procedure.

The DATA command creates files with variable length, blocked records, with a block size of 2048 bytes.

ISAM file records are prefixed with an 8-digit key incremented from the right in increments of 100 beginning at zero. For example, the first record would be 00000100, the second 00000200, etc.

DROP Command

DROP command is used to remove the specified file definition from hold status. (See examples 1 through 4 under the description of the HOLD command.) If an outstanding RELEASE command (macro) exists for the definition, normal release processing occurs (that is, the file definition is deleted and the appropriate private devices are released).

Name	Operation	Operand
	DROP	link-symbol

link-symbol

Specifies the file definition name.

Default: A link name of spaces is assumed.

END Command

END will terminate a file created by DATA command and cause the file to be closed.

Name	Operation	Operand
	END	

Programming Notes

During the spoolin process, an END command will be assumed to be missing if a LOGON card is encountered other than immediately after a DATA command. When a subsequent DATA command is encountered, an END command is assumed.

ERASE Command

The ERASE command is used to make a file logically empty, to deallocate the space assigned to a file, and/or to remove the file's entry from the catalog. This command processes partially qualified filenames.

Name	Operation	Operand
	{ERASE / ER}	{filename-name / $userid. / *} [,DATA|SPACE|CATALOG]

filename

Specifies the fully or partially qualified filename (up to 54 bytes). Only the controller may specify other than his own $userid.

$userid.

Specifies all files for the user. A user may only specify his own user identification. The controller may specify any user's identification.

*

Specifies the task's current (EAM) object module file.

Note: One of the above operands is mandatory in the ERASE command. The omission of the following optional operands causes space to be deallocated (SPACE) for the file and the file's entry removed (CATALOG) from the catalog entry table.

DATA

Specifies that the file or files be made logically empty. The file or files remain cataloged and the space remains allocated.

Note: The file entry is identical to one for a file which had a FILE command (and, optionally, a CATALOG command) applied to it, but had not yet been opened.

SPACE

Specifies that space assigned to a file is to be deallocated. The file remains cataloged, but is considered to be logically empty (that is, the DATA option is performed).

CATALOG

Specifies that the catalog entry for the file or files be removed. This parameter is not allowed for files residing (allocated) on PUBLIC volumes.

Programming Notes

1. A user can only ERASE files that he owns.

2. Note that this command requires that all direct-access private volumes associated with a single file must be concurrently mounted if space is being deallocated. If the CATALOG option is specified, the associated private volumes are not required. If the DATA option is specified, the first private volume associated with each file is required.

3. If a set of files is specified (by way of a partially qualified filename), it is not required to have volumes for all files in the set concurrently mounted. Requirements in the previous paragraph apply.

Example

Assume that file A.1 is contained in two private volumes and file A.2 is contained on three private volumes (see figure 3-1), then:

ERASE A.

requires three devices (not five) since A.2 takes 3 volumes and all of them must be mounted simultaneously. In other words, the number of devices necessary to ERASE a set of files is equal to the number of volumes of the file distributed on the largest number of volumes.

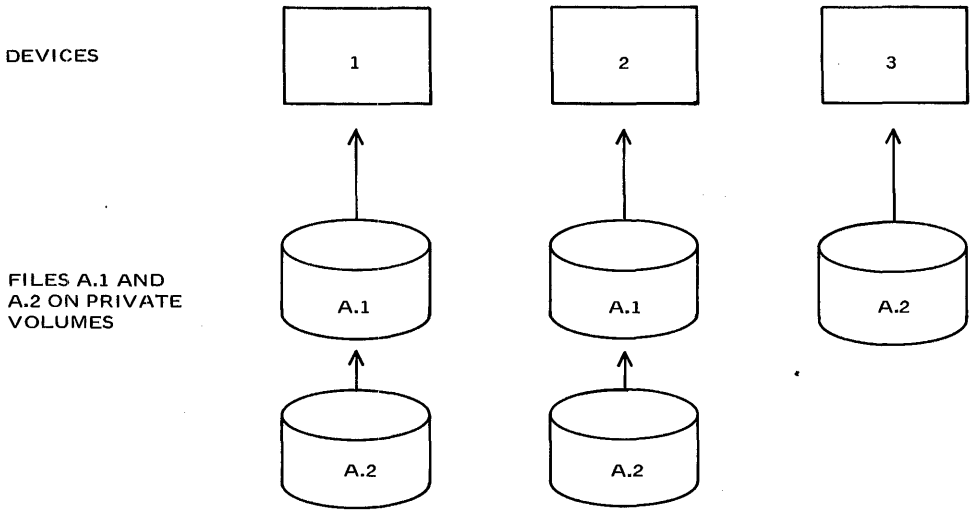


FIGURE 3-1. EXAMPLE — (ERASE COMMAND)

If a file is erased from a private volume, the device on which the volume resides will be acquired for the task. At completion of the ERASE process, the device will be returned to the system.

FILE Command/Macro

The FILE command is used to define a file and its characteristics to the system. Every file, except those processed by EAM, must be defined by this command. Typically, subsequent uses of a file do not require this command, since the file is cataloged and its properties are already known by the system.

Although this command has a multitude of parameters, the user need not specify anything more than the filename for public volume files. In many cases, the use of this ostensibly complex command is both simple and brief.

The FILE command performs the following major functions:

1. Catalogs a file
2. Allocates (and deallocates) random access space
3. Assigns devices
4. Alerts the operator to mount private volumes
5. Completes or modifies the FCB (File Control Block).

Name	Operation	Operand
	FILE	{ filename | *DUMMY } [,LINK=symbol]
		[,DEVICE=D564|D568|D590|TA|TAPE|T9N|T9P|T7cc|WORK]
		[,MOUNT= { volume sequence number-integer | (vol. seq. no,...) }]
		[,VOLUME= { PRIVATE | volume serial number-alphnum | (PRIVATE,number of volumes-integer) | (volume serial number-alphnum,...) }]
		[,SPACE= { primary-integer | ({ primary-integer[,secondary-integer] | first page-integer,amount-integer,ABS }) }]
		[,STATE= FOREIGN]
		[,FCBTYPE=SAM|ISAM|PAM|BTAM]
		[,RECFORM= { ([F|V|U],[A|M|N]) | F|V|U }]
		[,BLKSIZE=buffer size - { STD | (STD,integer) | integer }]
		[,RECSIZE=record size-integer]
		[,KEYPOS=displacement-integer]
		[,OVERLAP=YES]
		[,KEYLEN=key length-integer]
		[,DUPEKY=YES]
		[,PAD=percent of space-integer]
		[,LABEL=STD|NSTD]
		[,RETPD=days-integer]

Name	Operation	Operand

[,OPEN=INPUT|OUTPUT|EXTEND|REVERSE|UPDATE|OUTIN|INOUT|SINOUT]

[,LOGLEN=logical flag length-integer]

[,VALLEN=value flag length-integer]

[,VALPROP=MAX|MIN]

[,DDEVICE=D564|D568|D590]

[,DSPACE= { primary-integer / (primary-integer[,secondary-integer]) }]

[,DVOLUME= { volume serial number-alphnum / (volume serial number-alphnum,...) }]

[,SHARUPD=YES|NO]

[TVSN= { volume serial number-alphnum / (volume serial number-alphnum,...) }]

filename

Specifies the fully qualified filename. If the file is not cataloged, it is cataloged as a nonshared file with read/write access.

*DUMMY

Specifies a special file. If one attempts to read this file, control is passed to the end of the file address specified in the exit list. If one attempts to write information in the file, the information is lost (an infinite "garbage pail"). This special file is useful during debugging and also provides a convenient way to ignore some particular output.

Note: If this parameter is specified, the other parameters in the command except LINK are totally ignored. Accordingly, no device allocation, storage allocation, or cataloging takes place for dummy files.

LINK=

Specifies that the file parameters given in this command are to be used to modify those in an FCB with the same linkname. Note that if this parameter is omitted, a linkname of spaces will be generated, and no FCB will be modified by this file command. Note that LINK=, is not allowed.

If a FILE command has the same linkname as a previous file definition, the new command replaces the old definition. The processing is equivalent to the sequence:

File Command I	FILE LINK = x RELEASE LINK = x, KEEP
File Command II	FILE LINK = x

Notes:

1. Any private devices associated with FILE Command I are returned to the task, not the system. This important property allows one to reuse private devices.

2. The devices associated with FILE Command I are not attached (associated) with FILE Command II; they are merely returned to the task.

3. The LINK name is the entry that establishes the connection between the FCB and the DMS Job Control language.

CAUTION: A FILE command applies to every FCB with the same linkname, and can in fact apply to many FCB's concurrently. Note that the default linkname is "spaces" — not null and not ignored.

DEVICE=

Specifies the type of device.

D564

Specifies an 8564 Disc.

D568

Specifies an 8568 Mass Storage Unit.

D590

Specifies an 8590 Disc.

TA

Specifies a 9-level tape or a 7-level tape with data convert. The 7-level tape is assumed to have a control code (cc) of $(F0)_{16}$; that is, odd parity, pack/unpack, and no translate.

TAPE

Specifies any type of 9-level tape.

T9N

Specifies an NRZ (nonreturn to zero) 9-level tape.

T9P

Specifies a phase-modulated 9-level tape.

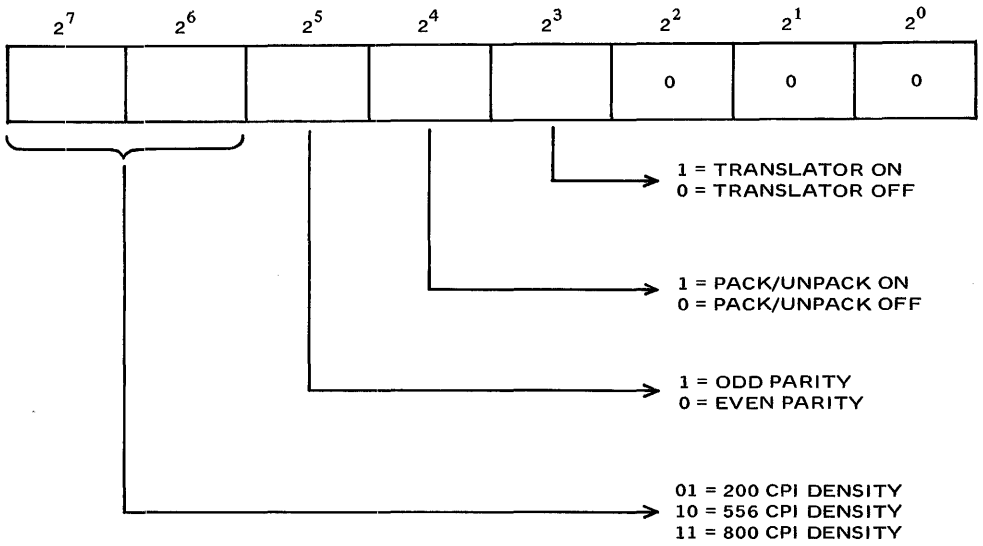
T7cc

Specifies a 7-level tape. The control code (cc) must specify the standard write control byte required by 7-level devices. See figure 3-2 for Control Byte and 7-level Control Codes.

WORK

Specifies a system work tape. The system attempts to acquire a 9-level system work tape previously specified by the operator (via the SETUP command). If no tape is available, the parameter is equivalent to the TAPE parameter. If this parameter is specified, the VOLUME parameter is ignored. Note that the RETPD parameter is also ignored and is assumed to be zero days.

Default: If no DEVICE parameter is specified, public volume devices are assumed.



THERE ARE 15 POSSIBLE 7-LEVEL MODES:

CONTROL BYTE VALUE (16)	DENSITY	PARITY	PACK/UNPACK MODE	TRANSLATE MODE
60	200	ODD	OFF	OFF
A0	556	ODD	OFF	OFF
E0	800	ODD	OFF	OFF
40	200	EVEN	OFF	OFF
80	556	EVEN	OFF	OFF
C0	800	EVEN	OFF	OFF
68	200	ODD	OFF	ON
A8	556	ODD	OFF	ON
E8	800	ODD	OFF	ON
48	200	EVEN	OFF	ON
88	556	EVEN	OFF	ON
C8	800	EVEN	OFF	ON
70	200	ODD	ON	OFF
B0	556	ODD	ON	OFF
F0	800	ODD	ON	OFF

FIGURE 3-2. CONTROL BYTE AND 7-LEVEL CONTROL CODES

MOUNT=

This parameter is only applicable to private volumes.

volume-sequence number

Specifies the volume sequence number of the volume or volumes to be mounted. If 0 (zero) is specified, no mounting is performed.

Default for Tape: First volume is mounted.

Default for Direct Access: All volumes containing the file are mounted.

The SPACE parameter is processed before this parameter. For direct-access files, this parameter applies only to volumes which actually have space allocated on them for the associated file at the time the parameter is processed.

Example 1:

Assume file JOE does not exist.

```
FILE JOE,SPACE=90,DEVICE=D564,VOLUME=(000123,
000456,000789),MOUNT=(1,2)
```

If all 90 pages can be allocated on volume 000123, the other two serial numbers are not placed in the catalog. The mount parameter is then considered erroneous, since volume sequence number 2 does not exist.

Example 2:

Assume a 21-page file XRAY exists on volume serial number 000123 and the user wishes to add an additional 90 pages to the file:

```
FILE XRAY,SPACE=90,VOLUME=(000456,000123),
DEVICE=D564
```

Also, assume that 30 of the additional pages requested are allocated on volume 000456 and the remaining 60 pages on volume 000123.

The catalog contains three volume entries:

entry 1 – VSN 000123 – 21 pages
entry 2 – VSN 000456 – 30 pages
entry 3 – VSN 000123 – 60 pages

Thus, volume sequence numbers 1, 2, and 3 address volumes 000123, 000456, and 000123, respectively. If the command is written as:

FILE XRAY,SPACE=90,DEVICE=D564,VOLUME=(000123, 000456)

and the first 60 additional pages are allocated on volume 000123 and the remaining 30 on 000456, the catalog would contain two volume entries:

entry 1 – VSN 000123 – 81 pages
entry 2 – VSN 000456 – 30 pages

For tape files, the MOUNT parameter applies to the combined set of volume serial numbers in the catalog prior to issuance of this command plus those specified in the VOLUME parameter.

VOLUME=

PRIVATE

Specifies that volumes are to be allocated from the operator's pool (for example, the tapes or disk available to the system operator). Once assigned, the volume belongs to the programmer.

number

Specifies the number of PRIVATE volumes desired ($1 \leqslant$ number $\leqslant 9$).

The notation VOLUME=(PRIVATE,2) does not necessarily mean that 2 volumes will be mounted. The appropriate MOUNT option must be used. In the example given, 2 volumes will be allocated from the operator's pool and a MOUNT=(1,2) must be used so that the two volumes are mounted on 2 drives. If the MOUNT is not used, only 1 volume will be mounted and only 1 device will be used, although 2 VSN's are known to the system.

vsn

Specifies the volume serial number identifying the volume.

Note: The reader should reference the description of the SPACE parameter to determine how vsn's for direct access volumes are entered into the system.

For a new tape file, the volume serial numbers are placed in the catalog in the order given. For an existing tape file, the volume serial numbers are added to those already in the catalog in the order given. Of course, these numbers cannot duplicate those already in the catalog.

SPACE=

Specifies the direct-access storage allocation for the file. (See examples under the MOUNT= parameter.) Although this parameter has several options, the parameter can be defaulted for the user creating a new file (whether public or private). Accordingly, the primary and secondary space allocations specified at system generation time are assigned.

primary

Specifies the number of half-pages (2048 bytes) to be allocated for primary allocation. This number is rounded up, if necessary, to a multiple of 3.

secondary

Specifies the number of half-pages (2048 bytes) to be allocated whenever the file's space is exhausted and more space is required. This number is rounded up, if necessary, to a multiple of 3. Negative values are not allowed.

first page

Specifies the physical half-page with which to begin allocation. (The first half-page of a device is 1.) The value must be 1, 4, etc. (that is, first page/3 yields a remainder of 1).

amount

Specifies the number of half-pages required. The value must be a multiple of three.

ABS

Specifies that absolute allocation is desired. If this is the initial allocation for the file, the secondary allocation value is set to 0.

Notes:

1. For a file for which space has already been allocated, primary may specify a negative number. This causes the system to deallocate space. In this case, the VOLUME parameter is ignored. The system deallocated the space beginning with the last volume specified in the catalog and deallocates until the specified amount is removed or until a half-page which contains data (applicable to ISAM and SAM) is encountered. In a sense, deallocation is an end-to-front process; that is, space is only removed from the end of the file's space.

2. It is permissible to change the secondary allocation specification for a file.

3. The space specified is for the total file; that is, it is not on a per volume basis. Moreover, if several volume serial numbers are specified in the VOLUME parameter for a file which is initially being allocated, the system attempts to obtain all the space on the first volume. If successful, the other vsn's are ignored and are not placed in the catalog.

4. If no vsn's are specified for a private volume file for which allocation already exists, space is first obtained on the last (most recently allocated) volume of the file. If sufficient space does not exist on this volume, the system attempts to allocate the space on the other volumes in the catalog. If vsn's are specified, the space is only obtained from the specified volumes as described above. If this space is insufficient, a partial allocation will result; that is, space existing on vsn's in the catalog but not specified in the current FILE command will not be utilized.

5. For a public volume file for which space already exists, space is first obtained on the last volume of the file. If the space available on that volume is insufficient, the system selects the public volume which has the most space available. (This is not exactly the case. The strategy is to select the volume which has the highest percentage of free packets; a packet is a set of 24 contiguous half-pages. In the appendices, see "Volume Table of Contents (VTOC) Formats," for a discussion of packets, etc.).

6. Only the controller may allocate on a specific public volume or volumes or on specific public devices. In this case, the space is only obtained from the specified volumes or device types. As before, if vsn's are specified, the system attempts to satisfy the space requirements on the first volume specified before inspecting any others.

7. Only the controller may allocate absolutely (ABS) on public volumes.

8. When allocating absolutely, either zero or all the space specified is obtained. Contrary to the other rules, if a vsn is specified, only the first vsn is used; that is, absolute allocation is on one volume per command.

9. The user can force the spreading of allocation of space over a set of n volumes by issuing this command n times.

10. It is permissible to allocate absolutely on one occasion for a file, and to subsequently allocate nonabsolutely (and vice versa).

11. Secondary allocation may be specified as 0 to prohibit dynamic allocation. Note that subsequent allocations only change the value for secondary when it is explicitly specified.

12. The primary allocation for a file may be specified as 0. This is meaningful, for example, where the user only wishes to change the value for secondary allocation and does not wish more space.

13. Except when allocating absolutely, the system obtains all the space requested or as much as possible. The typical obstacle to allocation is that the specified volumes do not have enough free space.

Note: The user cannot deallocate space not belonging to him. Only the controller can specify a filename which begins with the $ character if a negative SPACE value is given.

14. Deallocation of space from an ISAM file created with the index location option will result in the removal of unused pages from the data and/or index volumes depending on which volume extents were last obtained. Space cannot be selectively deallocated from a data or an index volume.

STATE=

FOREIGN

Specifies that an existing, uncataloged, private volume file is to be cataloged. Such a file might have been created on another VMOS system, or it might have been previously removed from the system (via the ERASE command). Note that a foreign direct-access file had to be created by version 6 or later versions of DMS (not, for example, by TOS).

For a tape file, all the vsn's of the file must be specified for the file, and they must be specified in proper order. It is not necessary to have the tapes mounted and/or to have the devices secured to accomplish this function.

The device must be specific, that is, T9N, not TA or TAPE.

If a foreign tape file has standard labels, the file characteristics are moved from the label to the catalog at OPEN time. If the foreign tape file has nonstandard labels, the user must supply (by way of the FILE command) the file characteristics normally contained in the HDR2 label. It is the user's responsibility to ensure that the linkname used in this FILE command is equal to the linkname in the FCB when the foreign file is opened, since the file characteristics are not entered into the catalog until OPEN time.

If this is a direct access file, the first vsn of the file is specified; only the first volume of the file need be mounted to accomplish this function.

Note that if one or more files which begin on this volume have the same name, even if the userids are different, the system will always assume that the first file physically found in the format-1 labels is the one to be used. A warning message is issued to note this condition. The user could subsequently remove the file (via the CATALOG command) to resolve the problem in this specific instance.

It is essential that a FOREIGN file being introduced into a new system from a private random access device be removed from the catalog of the system in which it is currently cataloged before FOREIGN volume processing is invoked. A FOREIGN file is always cataloged as a read-only file.

Example:

File A.A is currently cataloged in system TSOSRIV. This file is to be introduced into system TSOSEA as a FOREIGN file. An ERASE A.A, CATALOG must be performed under system TSOSRIV. After this is performed, A.A may be introduced into system TSOSEA as a FOREIGN file.

If the user wishes to catalog a single-volumed foreign tape file without specifying VSN, he may mount this volume on a secured tape drive and specify device=work. It is the user's responsibility to mount the right volume on the secured work drive. DMS will record the VSN and all other characteristics at open time. This facility is only applicable to single-volumed foreign tape files.

DDEVICE, DVOLUME, DSPACE, SHARUPD

The DDEVICE, DVOLUME, and DSPACE parameters are specified in the same manner as the corresponding DEVICE, VOLUME, and SPACE parameters with the exception that DDEVICE may only specify a random access device and no default is assumed for DSPACE for a file that has not been previously cataloged.

The DDEVICE, DVOLUME, and DSPACE parameters may only be specified for an ISAM file, and may only refer to private volumes. If the user requires any one of these parameters and if the file does not yet have any space allocated to it, all three must be specified. If space has been allocated for the file, a FILE command can be issued to extend this space, in which case the DSPACE parameter may occur without DDEVICE and DVOLUME parameters.

When the three parameters (DDEVICE, DVOLUME, or DSPACE) are specified for an ISAM file, the DEVICE, VOLUME, and SPACE parameters, whether specified or defaulted, refer to the index portion of the ISAM file while the DDEVICE, DVOLUME, and DSPACE parameters refer to the data portion of the ISAM file. In this way, it is possible to create an ISAM file which has, for example, all its data on 568 (MSU) volumes while all the index pages reside on a set of 564 or 590 volumes.

Once an ISAM file is created in this way, it is not possible to get all the index and data blocks on the same device type. The only way to achieve this is to copy the ISAM file to a second ISAM file whose space has been allocated without using the DDEVICE, DVOLUME, and DSPACE parameters.

All of the volumes of such a split ISAM file must be private. It is not possible to split an ISAM file between two different public device types or between public and private devices.

The SHARUPD parameter is used to indicate whether an ISAM file may be shared among tasks for the purpose of updating.

TVSN

Specifies the volumes to be processed in a given program. This parameter is only applicable to cataloged tape input files. When this parameter is specified, volume(s) which already exist in the catalog are temporarily ignored for this program, and volumes specified by this parameter will be used. Note that this does not alter the catalog. This command will be terminated if both volume and TVSN are specified.

Programming Notes

The remaining parameters, which are used to complete an FCB, are discussed in the FCB macro instruction. See VMOS General Service File Management Macros instructions.

Each file command/macro causes a TFT (Task File Table) to be created. Each TFT contains information about file characteristics and device(s) acquired to this file, and TFT's are identified by their link names. If another file command/macro is issued bearing the same linkname, the existing TFT will be deleted and all devices, if any, will be deleted. A new TFT will be created and new devices (if requested) will be acquired. In general, devices attached to the new TFT will not be the same as those possessed by the previous TFT. However, for a very special case, the devices will be retained. When the following conditions all prevail, the devices are retained:

1. The previous TFT is created for a Class-I program.

2. The new TFT to be created is a VMOS file.

3. The device must be in the tape family and the device type must be the same for the old and the new TFT's.

4. The number of devices requested in the new TFT must not exceed one.

FSTATUS Command

This command furnishes the status information for a file. The command processes partially qualified filenames.

Operation	Operand
{FSTATUS {FSTAT	[filename-name|$userid.] [,STANDARD|S] [,CATALOG|C] [,TRAITS|T] [,PASSWORD|P] [,ALL]

filename

Specifies the fully or partially qualified filename (up to 54 bytes).

$userid.

Specifies all files for the user. If a user specifies another user's identification, only information on shared files will be displayed.

Default: The $userid of the user logged on is assumed.

If no option parameters are specified, the filename(s) and file size are displayed.

The following parameters may be specified to obtain additional information:

STANDARD|S

1. Type of file (that is, the access method used to create the file).

2. Type of volumes on which the file resides (that is, public or private volumes).

3. Secondary allocation value for the file.

4. Highest page utilized by the file (ISAM files and SAM files with standard blocks).

CATALOG|C

Specifies that security-type information, typically entered via the CATALOG command, is to be displayed. This includes the following:

1. Sharability of the file.

2. Password requirements (but not the passwords themselves).

3. Access to the file (that is, READ only or READ and WRITE).

4. Creation date.

5. Expiration date.

6. Indicates whether FRS RECON and/or RESET services are being used. If the status of FRS services has been changed since the file was last accessed and at least one of the services is turned off, an asterisk will be placed following the NO condition. When both RECON and RESET equal NO and the asterisk is present, the user may reinitiate one or both of the services turned off without loss of previous FRS support.

TRAITS|T

Specifies that detailed file and volume characteristics are to be displayed, such as:

1. Record format and size.

2. Block size and type.

3. Key length and key position (ISAM files).

4. Volume serial number and device type of each volume on which the file resides.

5. Lengths of logic and value flag fields for ISAM files.

PASSWORD|P

Specifies that the passwords are to be presented. Only the controller is given the actual passwords. The response to the user is YES if passwords are used and NO if not. The user who has forgotten his password must request the controller to issue this command on his behalf.

The passwords are displayed in both character and hexadecimal form, since the password might contain unprintable characters.

ALL

Specifies that the information supplied by the STANDARD, CATALOG, and TRAITS parameters is to be displayed.

Programming Notes

Although the specification parameters are essentially considered as positional, they may be written in any order.

Example:

```
FSTATUS JOE,STANDARD,CATALOG
FSTATUS JOE,CATALOG,STANDARD
```

The above examples accomplish identical results.

The printing of STATUS information may be interrupted by depressing the Break key. A slash is displayed and the next command may be issued.

Example 1 illustrates use of the FSTATUS command for a password-protected File, TEXT. Example 2 illustrates the various options of the FSTATUS command.

Examples

Example 1: FSTATUS command for a password-protected file.

```
%B001 PLEASE LOGON.
/LOGON RINGO
%B002 LOGON ACCEPTED AT 1122 ON 11/16/69, TSN 0062 ASSIGNED.
/FSTATUS
%0000042 SAMFILE
%0000012 SAMPUT
%0000012 MAC3FILE
%0000042 ISAMFILE
%0000006 PAMFILE
/FILE TEST,SPACE=(12,3),DEVICE=D564,VOLUME=PUB200
/CAT TEST,STATE=UPDATE,RDPASS=1111,ACCESS=READ
/FSTATUS TEST,STANDARD,CATALOG
%FILE STATUS NOT GIVEN FOR:
%NON-SHARABLE FILES=00000      PASSWORD PROTECTED FILES=00001
/PASSWORD 1111
/FSTATUS TEST
%0000012 TEST
%TOTAL PUBLIC PAGES ALLOCATED = 0000012
/FSTATUS TEST,STANDARD,CATALOG
%0000012 TEST
%   FCBTYPE = NONE              VSN TYPE= PUB
%   2ND ALLO= 000               LAST PG = 0000000
%   SHARE   = NO                ACCESS  = READ
%   CR DATE = 00000             EX DATE = 00000
%   RDPASS  = YES               WRPASS  = NO
%TOTAL PUBLIC PAGES ALLOCATED = 00000012
/LOGOFF
%B003 LOGOFF AT 1126 ON 11/16/69, FOR TSN 0062.
%B014 CUP TIME USED: 0000.6173 SECONDS.P
```

Example 2: Various options of the FSTATUS Command.

```
%B001 PLEASE LOGON.
/LOGON RINGO
%B002 LOGON ACCEPTED AT 1002 ON 11/09/69, TSN 0228 ASSIGNED.
/FILE VER006
/FSTATUS VER006
%0000006 VER006 <------o default space allocation
/FILE VER006,SPACE=3
```

```
/FSTATUS VER006
%0000009 VER006 <-------o space increased
/FSTATUS VER006,STANDARD<-------------------------o3 FSTATUS
%0000009 VER006                                    options
%   FCBTYPE = NONE          VSN TYPE= PUB
%   2ND ALLO= 003           LAST PG = 0000000
/FSTATUS VER006,CATALOG <-------------------------|
%0000009 VER006                                   |
%   SHARE   = NO            ACCESS  = WRITE       |
%   CR DATE = 0000          EX CATE = 00000       |
%   RDPASS  = NO            WRPASS  = NO          |
/FSTATUS VER006,TRAITS  <-------------------------|
%0000009 VER006
%   RECFORM = NONE          RECSIZE = 00000
%   BLK TYPE= NONE          BLKSIZE = 000000
%   VSN/DEV = PUB25]/D564
/FSTATUS VER006,STANDARD,CATALOG,TRAITS<-----oall 3 options in
%0000009 VER006                                same command.
%   FCBTYPE = NONE          VSN TYPE= PUB      /FSTATUS VER006,
%   2ND ALLO= 003           LAST PG = 0000000  ALL yields the
%   SHARE   = NO            ACCESS  = WRITE    same output. A
%   CR DATE = 00000         EX DATE = 00000    short form for the
%   RDPASS  = NO            WRPASS  = NO       operands can also
%   RECFORM = NONE          RECSIZE = 00000    be used, e.g.
%   BLK TYPE= NONE          BLKSIZE = 000000   /FSTATUS VER006,
%   VSN/DEV = PUB251/D564                      S,C,T
/ERASE VER006
/FSTATUS VER006
 %D D533 NO SUCH FILE..<-oerror message after ERASE
/LOGOFF
%B003 LOGOFF AT 1006 ON 11/09/69, FOR TSN 0228.
%B014 CPU TIME USED: 0000.8251 SECONDS.
```

FSTAT Macro-Furnish File Status Information (S)

This macro is used to furnish status information for the specified file. The macro may also process partially qualified filenames.

Name	Operation	Operand
[symbol]	FSTAT	[filename-name|$userid.] ,area-addr[,size-value] [,SHORT|LONG] [,PASSWORD]

filename

Specifies the fully or partially qualified filename (up to 54 bytes).

$userid

Specifies all files for the user. A user may only specify his user identification code. The controller may specify any user's identification code.

area

Specifies the area (relexp) into which the information is to be placed.

size

Specifies the size (absexp) in bytes of the input area. If the area is too small, the macro is terminated without any movement of data.

Default: An area of 60 and 2048 bytes is assumed for SHORT and LONG respectively.

SHORT

Specifies that the fixed portion of the catalog entry (currently 60 bytes) is to be placed in the user's area.

LONG

Specifies that the complete catalog entry is to be placed in the user's area. (A complete entry is variable in size and has a maximum size of 2032 bytes.)

Default: SHORT.

PASSWORD

Specifies that the passwords are to be made available. Only the controller may specify this parameter.

Programming Notes

If a filename is fully qualified, the catalog entry (SHORT or LONG) for the file is moved to the specified area. The password fields are set to binary 0's, unless the controller has issued this macro and has specified the PASSWORD operand.

If the first operand specifies a partially qualified filename or the $userid, the output of the macro is a set of filenames. The format of the output entry is:

						01
Length	Filename	-----	Length	Filename	00	00
1 byte	≤44 bytes				1 byte	1 byte

Length specifies length of the corresponding filename (plus the length byte). The list is terminated by $(00)_{16}$. If the area cannot contain all of the filenames, the termination byte is suffixed with a $(01)_{16}$; otherwise, this suffix byte is set to $(00)_{16}$.

Example

04	JOE	OF	PAYROLL.MASTER	00	00

The following return codes are set in register 15.

00 – request successfully completed.

The codes for nonsuccessful execution are specified in the IDEMS macro.

HOLD Command

The HOLD command is used to set the file definition, specified by the linkname, to a hold status. Thus, if a subsequent RELEASE command (macro) is issued for the file definition, the release processing is deferred unitl the DROP command is issued.

Name	Operation	Operand
	HOLD	[link-symbol] [,TOS]

link

Specifies the file definition name.

Default: a linkname of spaces is assumed.

TOS

Specifies that the file definition is for a TOS file.

Programming Notes

If the specified file definition does not exist, one is created. It is only for this case that the TOS parameter need be specified. Note, therefore, that it is permissible to issue a HOLD command before issuing a FILE command for the specified file definition.

Examples

Example 1

HOLD X	File definition entry X is created. The Hold indicator is set.
FILE LINK=X	This command places information in entry X.
RELEASE X	The entry is not released since the HOLD indicator is set. The release indicator is set.
DROP X	The entry is now released.

Example 2

FILE LINK=X	File definition entry X is created.
HOLD X	The Hold indicator is set.
.	
.	
.	
DROP X	The Hold indicator is reset; since no release indicator has been set, no other action occurs.
.	
.	
.	
RELEASE X	File definition entry X is released.

Example 3

```
SECURE TAPE=1

FILE LINK=X,
   DEVICE=T9N,                File definition entry X is created. A
   VOLUME=000123              private tape device is acquired for vsn
.                             000123.
.
.
HOLD X                        The Hold indicator is set.
.
.
FILE LINK=X                   A new file definition entry is created even
                              though the Hold indicator is set. The Hold
                              indicator will now apply to the new definition
                              of X. The tape is returned to the task for
                              reuse.

FILE LINK=Q,
   DEVICE=T9N,                The tape is now assigned to this file
   VOLUME=000124              definition entry.
```

Example 4

```
FILE LINK=X
.
.
HOLD X

CHANGE X,Y                    The name of the entry is changed. The
                              Hold indicator applies to entry Y.

DROP X                        This is an error; there is no entry
.                             with a linkname of X.
.
.
RELEASE Y                     The entry is not released since the
.                             Hold indicator is set.
.
.
DROP Y                        Entry Y is now released; that is, the
                              old X is released.
```

LOG Macro

The user program informs the access method that it intends to modify a record using the LOG macro.

Name	Operation	Operand
[symbol]	LOG	{(fcb) / (1)}

fcb

Specifies the address of the FCB macro associated with the file.

The LOG macro should be issued before a logical record is modified, but after it has been retrieved by the access method.

If before images are not required for the file, the LOG macro functions as a no-op.

See Part 2, Section 3 of this manual for more information on the File Reconstruction System (FRS), with which this macro functions.

PASSWORD Command

This command supplies file passwords for a task.

Name	Operation	Operand
	PASSWORD	password-special|(password-special,...)

password

Specifies a password or list of passwords needed to access a file. The password must be a 1- to 4-byte decimal, hexadecimal, or character self-defining term.

Programming Notes

The command creates a list of passwords, which may be added to by subsequent password commands. When a password is required to access a file, this list is searched unless the password already exists in the FCB used to OPEN the file.

A password of X'00000000' is ignored.

RECON Command

This command requests the reconstruction (of a portion) of the named file.

Name	Operation	Operand
	RECON	filename,(range,...)

filename

Specifies the fully qualified filename of the file to be reconstructed. This file must have previously requested reconstruction services with a RECON parameter in the FRS command (see VMOS Systems Management Manual).

range

Specifies the range of logical PAM pages which contain the record(s) to be reconstructed. Any number of ranges can be specified, on any number of RECON commands. A range is specified as follows:

page1:page2

(All PAM pages between page1 and page2 will be scheduled for reconstruction.)

page1

(Only the stated PAM page will be scheduled for reconstruction.)

page1

(ALL PAM pages between page1 and the end of the file will be scheduled for reconstruction.)

The requested action will be delayed until the System Controller authorizes it.

See Part 2, Section 3 of this manual for additional information on the File Reconstruction System (FRS).

RECON Macro

This macro is the user program equivalent to the RECON command, described more fully under DMS Commands.

Name	Operation	Operand
[symbol]	RECON	{fcb / (1)}

fcb

Specifies the address of the FCB macro associated with the file.

If FRS services for this file have not been requested by the System Controller, this action macro functions as a no-op.

See Part 2, Section 3 of this manual for more information on the File Reconstruction System (FRS), with which this macro functions.

RELEASE Command
REL Macro

Both the RELEASE command or REL macro delete the specified file definition. Any private devices associated with the file are optionally released and become available to the system for reassignment.

Name	Operation	Operand
	{RELEASE / REL}	[link-symbol] [,KEEP]

link-symbol

Specifies the linkname of the file definition.

Default: a linkname of spaces is assumed.

KEEP

Specifies that any private devices associated with the file are not to be returned to the system; that is, they are available for reuse by this task.

Programming Notes

If there is more than one active file on the private volume being released, the device will not be actually released until a RELEASE command is issued for every file on that volume.

The RELEASE command is ignored if the specified file definition is in hold status (reference HOLD command). The release processing will be activated, however, when the corresponding DROP command is issued.

The RELEASE command causes all tape volumes contained in the Task File Table to be logically removed from the system.

When the macro form is used, the return codes are set in register 15. 00 indicates successful completion of the operation. The codes for unsuccessful completion are specified in the IDEMS macro. The use of this macro is described in "File Management Error Message Concepts and Code Structure" in the appendices.

RESET Command

This command requests the resetting of the named file.

Name	Operation	Operand
	RESET	filename
		[,TIME=(date[,time])]
		[,COUNT= {integer / 1}]

filename

Specifies the fully qualified filename of the file to be reset. This file must have previously requested reset services with a RESET parameter in the FRS command (see VMOS System Management Manual).

TIME=

date

The date to which the file will be reset, in the form MMDD, where

MM = the month of the year, 01-12
DD = the day of the month, 01-31

time

The time of the above date, in the form HHMM, where

HH = the hour of the day, 00-23 (01=1 A.M.)
MM = the minute of the hour, 00-59

Notes

1. There is no default for the entire TIME parameter. If omitted, the COUNT parameter is used.

2. The '(date,time)' unit corresponds to the date/time stamp of the earliest record of interest. This need not be a date/time when the file was actually OPEN. It is interpreted to mean the date/time stamp of the first OPEN prior to '(date,time)', if the file was then OPEN.

COUNT=

Specifies the number of OPENs back to which the file is to be reset.

Default: 1, (that is, the file will be reset 1 OPEN.)

Note that if both COUNT and TIME are specified, the first one satisfied will complete the request. The requested action will be delayed until the system controller authorizes it.

Reset requests via the RESET command lock out any use of the file until the request is satisfied. Since this command (like the RECON command) can only be issued by the owner of the file, this ensures not only the integrity of the data in his file, but the integrity of reports generated by others from his data.

A RESET command overrides any previously unsatisfied RESET command. A RESET command with COUNT=0 will effectively delete any outstanding reset request and will immediately unlock the file unless there are Recons outstanding and the file is in allow status.

The number of OPENs saved by FRS is actually one greater than the count specified in the FRS command. The user is cautioned that OPEN INPUT and REVERSE do not count for FRS purposes; they are not logged. Multiple OPENs (using the ISAM Shared Update facility) count as one OPEN for FRS RESET processing.

SYSFILE Command

SYSFILE allows the user to reassign the SYSIPT, SYSDTA, or the TASKLIB files to another file.

Name	Operation	Operand
	SYSFILE	{SYSIPT / SYSDTA} = {filename / (SYSCMD) / (mn) / (CARD) / (PRIMARY)}
	SYSFILE	TASKLIB = {filename / (NO)}

filename

Identifies the name of the cataloged file that is to replace the named file. (The command is invalid if the filename has not been cataloged.)

For SYSIPT and SYSDTA filename or any one of the following operands is permitted.

(SYSCMD)

Changes the SYSIPT or SYSDTA file to the same file to which SYSCMD is currently assigned.

(mn)

Specifies the 2-byte installation mnemonic of the device to which SYSIPT or SYSDTA is to be assigned.

(CARD)

Indicates that SYSIPT or SYSDTA is to be assigned to the card reader.

(PRIMARY)

Indicates that SYSIPT or SYSDTA is to be changed to its primary assignment.

TASKLIB

Refers to the name of a program library file to be used by the Linking Loader or Linkage Editor. A (NO) option removes the previous task library filename.

Examples

```
/.NAME SYSFILE     SYSIPT=AFILE849.MASTIN     ①

/SYSFILE    SYSIPT=(SYSCMD)                   ②

/SYSFILE SYSIPT=(R1)                          ③

/SYSFILE SYSDTA=(PRIMARY)                     ④
```

① The user has specified that the SYSIPT is to be replaced by the cataloged file whose filename is AFILE849.MASTIN.

② The user requests that the SYSIPT be assigned to the same file to which SYSCMD is currently assigned.

③ The user requests that the SYSIPT device be made the device whose installation mnemonic is R1.

④ The user requests that the SYSDTA file be changed to its primary assignment.

Note: If the parentheses are not present when SYSCMD, PRIMARY, or CARD are specified, the system will consider them as file names.

ACCESS METHOD DEFINITION MACROS SUMMARIES

Access method macros constitute the VMOS instruction facilities which enable the VMOS user to define the format of his data. The following summaries classify these macros by access method. The programmer is cautioned that although several macros have the same name, their use, with the exception of OPEN, is unique to a particular access method; that is, each is restricted to the access method with which it is identified.

Basic Tape Access Method (BTAM) Macros

Macro	Function
BTAM	Allow the user to request BTAM action to read, write, check, wait control, or position tape files.
FCB	Define BTAM File Control Block.
OPEN	Ensure that the BTAM access routine is loaded and that the address relations are completed. Refer to "VMOS General Service File Management Macros" in the appendices for the description of this macro.

Evanescent Access Method (EAM) Macros

Macro	Function
EAM	Allow the user to request EAM action to open, read, write, check, erase, close, or wait temporary files.
FCB	Define EAM file control block.

Indexed Sequential Access Method (ISAM) Macros

Macro	Function
ELIM	Eliminate a record from a file.
FCB	Define ISAM file control block.
GET	Retrieve the next logical record in the file in ascending sequential order by record key.
GETFL	Retrieve the next logical record satisfying the flag criteria.
GETKY	Retrieve a logical record having a specified record key.
GETR	Retrieve the next logical record in a file in descending sequential order by record key.
INSRT	Place a new record in a file in the position determined by its record key.
OPEN	Ensure that the ISAM access routine is loaded and that the address relations are completed. Refer to "VMOS General Service File Management Macros" in the appendices for the description of this macro.

Macro	Function
PUT	Add a logical record to a file. PUT is normally used for initial file creation by recording logical records in ascending sequential order by record key.
PUTX	Replace a record retrieved by GET, GETFL, GETKY, or GETR.
SETL	Specify the location in the file from which subsequent processing is to take place: the beginning of the file, the end of the file, or at the location of a record with a designated key.
STORE	Place a record in a file at the position determined by the record's record key.

Primitive Access Method (PAM) Macros

Macro	Function
FCB	Define a PAM file control block.
OPEN	Ensure that the PAM access routine is loaded and that the address relations are completed. Refer to "VMOS General Service File Management Macros" in the appendices for the description of this macro.

Sequential Access Method (SAM) Macros

Macro	Function
FCB	Define SAM file control block.
FEOV	Advance the system to the next (tape) volume of a file before the end of the current volume is reached.
GET	Retrieve the next record from a file in physically sequential order.
OPEN	Ensure that the SAM access routine is loaded and that the address relations are completed. Refer to "VMOS General Service File Management Macros" in the appendices for the description of this macro.
PUT	Place a logical record on the file.
PUTX	Return an updated logical record to the file (direct-access volumes only).
RELSE	Cause any remaining logical records in a buffer to be bypassed for input. For output, the next logical record created is written as the first record of a new buffer.
SETL	Specify the position from which subsequent file processing is to take place.

The following material describes each of the access method definition macros. The descriptions include instruction format, programming considerations, and, where illustrative information is necessary, examples of the use of the instruction. The macros are presented in alphabetical order by access method.

BTAM Action Macro

BTAM – Read, Write, Check, Wait, Control, or Position(s).

All user requests for BTAM action are made via this macro.

Name	Operation	Operand
[symbol]	BTAM	fcbname-relexp [,operation-RD|RDWT|REV|REVWT| WRT|WRTWT|WT|CHK|FSF|BSF|WTM| RUN|ERG|FSR|BSR|REW] [,LOC= 1|2|relexp] [,LEN=absexp]

fcname-relexp

Specifies the address of the FCB associated with the file.

operation

Specifies the tape operation to be performed:

RD

Forward reading of a magnetic tape.

RDWT

Forward reading of a magnetic tape and wait until the operation is completed before returning control to the program.

REV

Reverse reading of a magnetic tape.

REVWT

Reverse reading of a magnetic tape and wait until the operation is completed before returning control to the program.

WRT

Writing of a magnetic tape.

WRTWT

Writing of a magnetic tape and wait until the operation is completed before returning control to the program.

WT

Wait for the previous I/O operation to terminate. Control is not returned to the user program until the operation has been completed and any necessary error-recovery functions have been performed.

CHK

Check for termination of previous I/O. If the I/O operation has not terminated, control is passed to the address specified in LOC. Otherwise, the operation is equivalent to WT.

FSF

Unwind the tape one tape mark.

BSF

Rewind the tape one tape mark.

WTM

Write a tape mark.

RUN

Rewind and unload a tape.

ERG

Cause an erase gap to be executed.

FSR

Unwind the tape one block.

BSR

Rewind the tape one block.

REW

Rewind the tape to BT.

Default: RDWT

Note: The Operations FSF, BSF, WTM, RUN, ERG, FSR, BSR, and REW always return the 5 status bytes in the FCB. The other operations only return these bytes if control is transferred to the ERRADDR EXLST address.

LOC=

Typically specifies the area to be read or written.

1

Specifies IOAREA1 address in the FCB.

2

Specifies IOAREA2 address in the FCB.

relexp:

Specifies a designated I/O area other than those given in the FCB.

If the operation is CHK, the LOC parameter must be in the form LOC= relexp. Control is passed to the location specified in LOC if the "checked" operation has not yet terminated.

Default: The IOAREA address in the FCB is chosen as follows: If the last address utilized was IOAREA1, then IOAREA2 is used, and vice versa. Of course, if the IOAREA2 area does not exist, IOAREA1 is used each time. The system maintains a field in the FCB which specifies the last IOAREA used.

LEN=

Specifies the number of bytes to be transmitted. If the parameter is omitted, the system obtains the count from the BLKSIZE parameter for format-F records, and from the register specified in RECSIZE for format-U records.

Programming Notes

1. Misuse of FSF or FSR can cause the tape to run away.

2. For any unsuccessful entrance to BTAM, control will be returned to the routine specified in the EXIT parameter of the FCB, with a code stored in the FCB. The hexadecimal codes are listed in "File Management Error Message Concepts and Code Structure" of the appendices (refer to the IDEMS macro).

Note that if the user specifies format-F records, but reads records not of the specified length, control is transferred to the ERRADDR EXLST address. If the actual record exceeds the specified size, an abnormal termination bit is set in Executive flag byte and the sense byte "data greater than count" is set. If the actual record is less than the specified size, an abnormal termination is set in the Executive flag byte; however, no sense byte or residual count is returned.

3. It is not necessary for the user to issue waits explicitly. BTAM will automatically do a wait before any read or write. If however, an error occurs in this wait, the return code will be 0C97 instead of 0C98 and the current read or write will not be executed.

4. Note that BTAM does not take an EOFADDR EXLST (end of file) exit. If a tape mark is read by RD, RDWT, REV, or REVWT, control is transferred to the ERRADDR EXLST address. Similarly, control is transferred to ERRADDR if one attempts to read reverse a tape already at BT.

5. If REV or REVWT is the operation mode, the first byte into which data is read is the address specified in LOC (or pointed to by LOC) + LEN -1.

6. The SAM macro FEOV can be issued for the file opened by BTAM.

7. In BTAM, record size may not exceed 4096 bytes.

8. In the processing of multiple tape volumes, no more than two tape units may be used.

FCB Macro Format, BTAM

Define An FCB For BTAM (0)

The following format indicates the parameter which may be supplied in the FCB macro instruction, when BTAM is being used. These parameters are described in detail in "VMOS General Service File Management Macros" of the appendices.

Name	Operation	Operand
[symbol]	FCB	FCBTYPE=BTAM [,LINK=linkname-symbol] [,FILE=filename-name] [,PASS=password-absexp] [,RETPD=retention period-absexp] [,RECFORM=(F|U|V,A|M|N)F|U|V] [,RECSIZE=absexp] [,BLKSIZE=absexp] [,IOAREA1=NO|relexp] [,IOAREA2=NO|relexp] [,EXIT=(relexp)|relexp] [,LABEL=STD|NSTD] [,OPEN=INPUT|REVERSE|OUTPUT|INOUT SINOUT|OUTIN]

EAM Macro

EAM Linkage

There is only one SVC with which to invoke EAM. The specific function to be performed is defined by the operation code in the Miniature File Control Block (MFCB). The interface macro, which is of type R, is:

Name	Operation	Operand
[symbol]	EAM	{MFCB-address-addrx (1)}

MFCB

Specifies the address of the MFCB (miniature file control block). This table, which is 20 bytes in length and must begin on a word boundary, is the user's medium of communication with EAM. The table format is listed in table 3-1 and described below.

TABLE 3-1. MFCB TABLE DEFINITION

Field Number	Field Description	Field Size (in Bytes)
1	Operation Code	1
2	Option Code	1
3	Logical Block Number	2
4	Filename	2
5	Error Byte	1
6	Status Bytes	5
	Unused	1
7	Address of I/O Area 1	3
	Unused	1
8	Address of I/O Area 2	3
	Total	20

Field Descriptions of the EAM MFCB Table

Operation Code

This field specifies the operation or function which EAM is to perform. Table 3-2 lists the OP codes and the fields of the MFCB which are read (R), set (S), or ignored (I). Before the functions of the operation code are described, the remaining fields are defined.

TABLE 3-2. MFCB OPERATION CODES VERSUS FIELDS USED

Operation Code	Operation Code in Hexadecimal	Fields used from MFCB						
		OPTION	BLOCK	ERROR	NAME	STATUS	AREA1	AREA2
OPEN	00	R	I*	S	S*	I	R	R
REOPEN	01	R	S	S	R*	I	R	R
READ	02	R	R	S	R	S	I	I
WRITE	03	R	R	S	R	S	I	I
CHECK	04	I	I	S	R	S	I	I
ERASE	05	I	I	S	R	I	I	I
CLOSE	06	I	S	S	R	S	I	I
rsvd	07							
CHECKW	08	I	I	S	R	S	I	I
rsvd	09							

*Note exceptions for the object module file described under programming notes.

Functions of EAM Operation Codes

OPEN (00)

Specifies that a new file is being defined. EAM assigns a filename (1 through 1023), creates certain entries in its own tables and retrieves and validates the I/O addresses and the option byte. Note the exceptions for object module files described in the programming notes.

REOPEN (01)

Specifies that an existing, closed, file is to be opened. The I/O address and the option byte are reprocessed at this time. Note that the logical block number field can be set to point at either the beginning or the end of the file.

READ (02)

Specifies that a block, as designated by the logical block number, is to be read. The block number may be 0 to n, where n is the last block of the file. Note that files may be read randomly. If the block number is 0, the next block of the file is read.

Example

Assume that the following read commands with the indicated block number are specified:

```
READ    1
READ    5
READ    0
READ    0
READ    2
READ    0.
```

BLOCKS 1, 5, 6, 7, 2, 3 would be read.

WRITE (03)

Specifies that a block, as designated by the logical block number, is to be written. The block number may be 0 to n+1, where n is the last block of the file. Note that existing blocks may be replaced, and new blocks can be added to the end of the file. If the block number is 0, a block is added to the end of the file.

Examples

Example 1

		Action
WRITE	1	Create block 1
WRITE	2	Create block 2
WRITE	3	Create block 3
WRITE	2	Replace block 2
WRITE	0	Create block 4
WRITE	0	Create block 5
WRITE	5	Replace block 5

Example 2

WRITE	1	Create block 1
WRITE	3	Error, EAM must create (add) new blocks without gaps or holes.

Example 3

WRITE	0	Creates a file of n blocks;
WRITEn	0	Note that the user need not bother with block numbers to create a sequential file.

Example 4

WRITE	1	Create block 1
WRITE	2	Create block 2
WRITE	3	Create block 3
READ	1	Read block 1
WRITE	0	Create block 4
READ	0	Error, end of file.

CHECK (04)

If the last operation was a READ or a WRITE, the status bytes from the CCB are placed in the status area. If the I/O which is being checked has not terminated, control is returned to the user and register 15 is set to $(00000008)_{16}$. If the I/O has terminated, this operation is identical with CHECKW.

CHECKW (08)

If the previous call was a READ or WRITE, the status bytes are stored in the caller's MFCB. If the I/O which is being CHECKWed has not terminated, the caller is pended until the I/O is complete. (See programming notes.)

ERASE (05)

The file is erased (destroyed) whether it is open or not.

CLOSE(06)

The file is closed, but it is not erased (destroyed). The highest block number of the file is placed in the Logical Block Number field in the MFCB.

Option Code

2^0 – this bit is inspected during READ and WRITE operations.

2^0 = 0 specifies that I/O area 1 is to be used.

2^0 = 1 specifies that I/O area 2 is to be used.

2^1 – this bit is inspected during REOPEN operation.

2^1 = 1 specifies that the value (0000) is to be placed in the logical block number field. EAM sets its internal block pointer to the beginning of the file.

2^1 = 0 specifies that the highest block number of the file is to be placed in the logical block number field. EAM sets its internal block pointer to the end of the file.

2^2 – 2^3 – reserved.

2^4 – this bit is inspected during the OPEN and REOPEN operations.

2^4 = 1 specifies that the task's object module file is being opened.

2^5 – this bit is inspected during OPEN and REOPEN operations issued by privileged programs.

2^6 – this bit is inspected during OPEN and REOPEN operations issued by privileged programs.

Logical Block Number

Specifies the 2-byte binary number of the block to be processed when a READ or WRITE op-code is invoked. The block number can range from 0 to 65,535. A block is the unit of information read or written; each block must be 2048 bytes in size. Accordingly, the user regards an EAM file as one or more blocks; if the block has a substructure, (for example, format V-records), this structure is independent of and unknown to EAM.

File name

Specifies the 2-byte filename, assigned (and placed in this field) by EAM at OPEN time. A task can have any number of EAM files. The total number of EAM files in the system (that is, for all tasks) cannot exceed 1023.

Error Byte

Specifies why the operation is unsuccessful. If an operation is successful, register 15 is set to $(00\ 00\ 00\ 00)_{16}$ and the error byte field is not set. If an operation is not successful, register 15 is set to $(00\ 00\ 00\ 04)_{16}$ and the field stipulates the error as follows:

2^0 Illegal operation

1. Operation code is invalid.

2. File is not open (READ, WRITE, CLOSE).

3. MFCB does not begin on word boundary.

2^1 Illegal filename

1. Filename = 0 (excludes OMF file) or filename greater than or equal to 1024 (excludes OPEN).

2. File has not been created.

3. File does not belong to the caller.

2^2 Illegal block number

1. Read is attempted to an empty file and logical block number is not zero (READ).

2. Update read or write is attempted and logical block number exceeds the current highest block number of the file (READ, WRITE).

2^3 Illegal I/O address

2^4 EAM space is not available.

1. No space has been allocated to EAM by the system controller (OPEN operation only).

2. The number of existing EAM files has already reached the limit of 1023 (OPEN).

3. Updating of the Saved File Index (SFI) is not successful for save option user (CLOSE, ERASE). Note that the basic function of CLOSE/ERASE has been completed.

2^5 Nonprivileged program attempts to process a privileged file.

2^6 End of file

1. An attempt has been made to read a nonexistent block. (Only when logical block number = 0 in MFCB).

2. Master EAM file – (SYSTFL) cannot be opened (OPEN).

2^7 I/O operation is not successful (READ, WRITE).

1. Hardware I/O error. The termination bytes in the status area should be inspected to determine the cause of I/O error which occurred on the previous READ or WRITE.

2. Software I/O error.

3. I/O initiation error, the last two bytes of the five status bytes contain PAM initiation error code, while the first three bytes are set to 'FF'.

Status Area

Contains five bytes from the CCB which EAM used to process the file. These bytes are CCB + 31 - 35, the Standard Device Byte, three Sense Bytes, and the Executive Flag Byte. (Refer to "Device Management Control Tables" in the appendices.)

The field is set only if (1) the last operation code was a READ or WRITE, (2) the current operation code is valid, (3) the current operation code is READ, WRITE, CHECK, CHECKW, or CLOSE, and (4) unsuccessful I/O operation.

Address of I/O Area No. 1

Specifies the virtual address of the first position of this I/O area. For READ operations, a block is read into this area, and for WRITE operations, a block is written from this area. EAM requires that the specified address and the specified address plus 2047 be in the same page.

Address of I/O Area No. 2

The description is the same as that of area No. 1. The two areas allow for buffering or file processing simultaneity.

Note: These I/O addresses are retrieved and validated at OPEN and REOPEN time only. The I/O addresses may be the same. Thus, the program does not need to have two I/O areas. This, of course, prohibits I/O overlap (buffering).

Programming Notes

When EAM receives a READ or WRITE operation, the user is given control as soon as the operation is accepted, which clearly implies that the I/O operation has not yet been completed.

Typically, a user would issue a READ followed (at the appropriate time) by a CHECKW. This guarantees that the I/O operation was completed. If, at the time CHECKW is issued, the last operation is not complete, the task is pended (waited). CHECKW also sets the status area in the MFCB.

This conventional processing works in EAM. A gross process chart for efficient reading of an EAM file is sketched.

```
SET I=1
OPEN
READ BLOCK I     (Sort of preread)
CHECKW

STEP 1
READ BLOCK I
PROCESS BLOCK I-1
CHECKW  (Overlap)
RETURN (to STEP 1)
```

In EAM, CHECKW is not required. It is desirable to avoid CHECKW because it requires an additional SVC. The CHECKW is obviated because READ is in reality implemented as a CHECKW and READ; similarly, WRITE is in reality CHECKW and WRITE. Note that with this type of implementation for READ and WRITE, if an I/O error occurs (2^7=1 in the Error byte), it applies to the previous I/O operation; the specified I/O operation is not performed.

Thus, the above process chart could be sketched as:

```
SET I=1
OPEN
READ BLOCK I

STEP 1
READ BLOCK I
PROCESS BLOCK I-1
RETURN (to STEP 1)
```

If the task's object module file is OPENed (2^4=1 in option byte), one of the following actions occurs:

1. If the object module file does not exist, then one is created. Its name (2-byte binary number) is placed in the MFCB; the logical block number field in the MFCB is set to binary zeros.

2. If the object module file exists, its name is placed in the MFCB; the highest block number of the file is placed in the Logical Block Number field in the MFCB.

If the task's object module file is REOPENed, then one of the following actions occurs:

1. If the object module file does not exist, the command is aborted; the 2^1 bit in the error byte field of the MFCB is set.

2. If the object module file exists, its name is placed in the MFCB; the highest block number of the file is placed in the logical block field of the MFCB.

When a file is created (that is, first OPENed) the leftmost four bits in the option byte, the I/O area No. 1 address and I/O area No. 2 address specified in the MFCB are saved in the system memory. To change these fields, it is necessary to CLOSE and REOPEN the file. Changing them in the program's MFCB is not sufficient.

GET – Get a Record (R) – Macro

The GET macro instruction retrieves the next sequential record of a file.

Name	Operation	Operand
[symbol]	GET	{fcb - addrx / (1)} [,{area - addrx / (0)}] [,{LOCK / NOLOCK}]

fcb

Specifies the address of the FCB associated with the file.

area

Specifies the address of the area into which the record is moved.

LOCK

Indicates that the data block containing this record is to be locked when the record has been retrieved.

NOLOCK

Indicates that the data block containing this record is not to be locked following execution of this macro.

Programming Notes

The area parameter is ignored if the IOREG parameter was specified in the FCB. If a record beyond the end of the file is requested, the user is given control at the EOFADDR (see EXLST macro). If the file contains duplicate keys, they will be retrieved in first-in, first-out (FIFO) sequence.

GETR – Get a Record Reverse (R) – Macro

The GETR macro instruction retrieves the next record in the file in reverse order (that is, toward the beginning of the file).

Name	Operation	Operand
[symbol]	GETR	{fcb-addrx / (1)} [,{area-addrx / (0)}] [,{LOCK / NOLOCK}]

fcb

Specifies the address of the FCB associated with the file being processed.

area

Specifies the address of the area into which the record is moved.

LOCK

Indicates that the data block containing this record is to be locked when the record has been retrieved.

NOLOCK

Indicates that the data block containing this record is not to be locked following execution of this macro.

Programming Notes

The area parameter is ignored if the IOREG parameter was specified in the FCB. If a record beyond the beginning of the file is requested, the user is given control at the EOFADDR address (see EXLST macro).

The program may switch from GET to GETR and vice versa at any time. There is no implication that it is necessary to position at either end of the file before changing direction. If a record with a key Kn is obtained by using a GET macro, and if the next macro issued is a GETR, then a record with a key of Kn-1 would be obtained where the following relationship holds:

$Kn\text{-}1 \leqslant Kn$

If the file contains duplicate keys, then the duplicates will be retrieved in a sequence of "last in first out" (LIFO) when the GETR macro is issued.

GETKY – Get a Record with Specified Key (R) – Macro

The GETKY macro instruction randomly retrieves a record with the specified key. Prior to execution of the macro instruction, the key of the record must be stored at the address specified in the KEYARG operand of the FCB.

Name	Operation	Operand
[symbol]	GETKY	{fcb-addrx / (1)} [,{area - addrx / (0)}] [,{LOCK / UNLOCK}]

fcb

Specifies the address of the FCB associated with the file being processed.

area

Specifies the address of the user's work area into which the record is moved.

LOCK

Indicates that the data block containing this record is to be locked when the record has been retrieved.

NOLOCK

Indicates that the data block containing this record is not to be locked following execution of this macro.

Default: locate mode is assumed. This macro instruction places the address of the record in the register specified by the IOREG operand.

Programming Notes

The area parameter is ignored if the IOREG parameter was specified in the FCB. If a record with the specified key is not found within the file, the user is given control at the NOFIND address (see EXLST macro).

After a GETKY macro is issued, the position indicator for GET and GETR are set up as if a GET macro had been issued to obtain the record. If the GETKY is unsuccessful, then a subsequent GET or GETR will obtain a record whose key is higher or lower, respectively, than the "not found" key. If duplicate keys exist in a file, then a GETKY macro will retrieve the record that was first placed in the file.

PUT – Write a Record (R) – Macro

The PUT macro instruction presents a logical record to the system to be included in the output file.

Name	Operation	Operand
[symbol]	PUT	{fcb-addrx / (1)} [,{area-addrx / (0)}]

fcb

Specifies the address of the FCB associated with the file being processed.

area

Specifies the address of the logical record to be moved into the output buffer.

Programming Notes

The area parameter is ignored if the IOREG parameter (that is, the locate mode) was specified in the FCB. The PUT action macro may only be used to add records to the end of a file. If a file is opened INOUT or OUTIN and the last action directed to the file was not a PUT, then PUT performs a SETL to the end of the file before putting the record.

Issuing a PUT in locate mode results in two actions: (1) the previous record built in the buffer is validated and actually added to the file, and (2) the address at which the next record is to be built is placed in IOREG. Since a valid IOREG value must be obtained before a record can be constructed, a dummy PUT must be issued prior to building the first record in the file.

Also, a valid record must be built after the last PUT prior to closing the file. Otherwise a CLOSE error exit will occur or an erroneous record will be appended to the file. For format-V records the value of the FCB field D1RECS1 must equal or exceed the size of the records PUT to the file. This can be accomplished either (1) by specifying the FCB parameter RECSIZE greater than or equal to the size of the largest record to be PUT or (2) by placing in the FCB field D1RECS1 before each PUT, the size of the next record to be built. (Note that if RECSIZE is unspecified it defaults to the value of BLKSIZE.)

This macro verifies that the key of any record transmitted is larger than or equal to the largest key currently in the file. If the monotonic sequence of keys is broken, the user is given control at the SEQCHK or DUPEKY address (see EXLST macro).

PUTX – Replace a Record in the File (R) – Macro

The PUTX macro instruction returns a logical record to the file. The record replaced is the last one retrieved by a GET, GETR, GETFL, or GETKY macro instruction.

Name	Operation	Operand
[symbol]	PUTX	{fcb-addrx / (1)} [,{area-addrx / (0)}]

fcb

Specifies the address of the FCB associated with the file.

area

Specifies the address of the logical record to be moved into the output buffer.

Programming Notes

The record key cannot be changed during the update process. If the key is changed, the user is given control at the SEQCHK address (see EXLST macro). If during an update process (in Locate mode) the program inadvertently modifies other portions of the data buffer, the results of subsequent processing will be unpredictable. If the data buffer is altered so that it can no longer be processed by ISAM, the user is given control at the USERERR address (see EXLST macro).

The length of variable format records may be modified only if the user is operating in Move mode.

No other ISAM action macro may be issued for the file between the PUTX and the relevant GET, GETKY, GETFL, or GETR macro.

If a PUTX is issued in Move Mode to a record within a sequence of records having duplicate keys, the position of the record within the sequence may be altered if the modification includes an increase to the original size.

INSRT – Insert a New Record (R) – Macro

The INSRT macro instruction obtains a logical record from the user's area and places it in the file in the position determined by the value of its record key.

Name	Operation	Operand
[symbol]	INSRT	{fcb-addrx / (1)} {,area-addrx / (0)}

fcb

Specifies the address of the FCB associated with the file.

area

Specifies the address of the logical record to be stored in the file.

Programming Notes

If a record with the same key value already exists in the file, the INSRT macro instruction will not record the new record (regardless of the DUPEKY specification in the FCB). The user is given control at the DUPEKY address (see EXLST macro).

STORE – Store a Record in the File (R) – Macro

The STORE macro instruction obtains a logical record from the user's area and places it in the file in the position determined by the value of its key.

Name	Operation	Operand
[symbol]	STORE	{fcb-addrx / (1)} {,area-addrx / (0)}

fcb

Specifies the address of the FCB associated with the file.

area

Specifies the address of the logical record to be stored in the file.

Programming Notes

STORE functions in the same manner as the INSRT macro instruction. If, however, a record is found in the file with the same record key, the new record is stored in place of the old. No continegency exit to the user is made. If DUPEKY=YES is specified in the FCB, duplicate keys are allowed. The new record is placed in the file immediately after any other records with the same key.

ELIM – Eliminate a Record in the File (R) – Macro

The ELIM macro eliminates records from a file.

Name	Operation	Operand
[symbol]	ELIM	$\left\{\begin{matrix}\text{fcb-addrx}\\(1)\end{matrix}\right\}\left[,\left\{\begin{matrix}\text{KEY}\\(0)\end{matrix}\right\}\right]$

fcb

Specifies the address of the FCB associated with the file.

KEY

Specifies that the key of the record to be eliminated will be found at the address specified by the KEYARG operand of the FCB. If this parameter is not specified, the last record retrieved by a GET, GETKY, GETFL, or GETR macro is eliminated. If 0 is written, the KEY code is placed into the register as follows:

Key Not Specified	Address of FCB
Key	Address of Key

Thus, if the register form is specified, and the programmer does not wish to point to a key, he places the address of his FCB in Register 0. If he does wish to specify a key, he places the address of this key in Register 0.

Programming Notes

No other ISAM action macro instruction can be executed between the relevant GET, GETR, GETFL, or GETKY macro instruction and the ELIM (NO KEY) action macro.

If the KEY parameter is specified and a record with the specified key cannot be found, control is returned to the user at the NOFIND address (see EXLST macro).

If duplicate keys exist in the file and if the KEY parameter is specified, the record which was first placed into the file is eliminated. Although the ELIM (NO KEY) must normally be preceded by a GET, GETR, GETFL, or GETKY, it may be executed for the first record in the file if it follows an OPEN and the state is INOUT.

The SETL macro instruction enables positioning to the beginning or end of a file or to any other location determined by a specified key.

Name	Operation	Operand
[symbol]	SETL	{fcb-addrx / (1)} [,{position-code / (0)}]

fcb

Specifies the address of the FCB associated with the file.

position

B

Specifies the beginning of the file.

E

Specifies the end of the file.

KEY

Specifies that positioning should be to the key specified through the KEYARG parameter of the FCB. If 0 is written, the position code is placed into the register as follows:

B	0
E	1
KEY	Address of KEYARG field in the FCB

Default: B.

Programming Notes

If the KEY position code is used and the specified key is nonexistent, positioning will still occur, and no error will result.

If duplicate keys exist within the file, then the position indicators will be set so that a GET macro will obtain the record which was first placed into the file.

The operation SETL B to a null file causes control to be sent to the EOFADDR EXLST address.

The GETFL macro-instruction is used to get the next record (in forward or reverse order) within specified limits whose flags satisfy the condition specified by the macro.

Name	Operation	Operand
[symbol]	GETFL	{fcb-addrx | (1)} [,{area-addrx | (0)}] [,VALTEST=(GT|GE|EQ|NE|LE|LT)] [,LOGTEST=(ANY|ALL)] [,LIMIT=(KEY,END)] [,REVERSE=YES] [,{LOCK | UNLOCK}]

fcb

Specifies the address of the FCB associated with the file being processed.

area

Specifies the address of the area into which the record is moved.

VALTEST=

If this parameter is specified, only those records whose value flag is in the given relationship to the information stored in the value flag portion of the area pointed to by KEYARG will be retrieved.

The meanings of the relationship symbols shown in parentheses are:

GT	Greater than
GE	Greater than or equal
EQ	Equal
NE	Not equal
LE	Less than or equal
LT	Less than

LOGTEST=

If this parameter is specified, only those records whose logical flag matches all or any of the bits of the mask stored in the logical flag portion of the area pointed to by KEYARG will be retrieved. If LOGTEST=ALL, a record is retrieved when all of the bits set in the mask have corresponding bits set in the record log field, though the record log field may also have other bits set in addition to those indicated by the mask.

LIMIT=

The scan for the next record starts at the current position of the file (as established by SETL or GETKY, for example) and proceeds to either the last record whose key is less than the key stored in the key portion of the area pointed to by KEYARG or the end of the file depending on whether KEY or END is specified. END is the default.

REVERSE=

If this parameter is specified the scan will occur in the reverse order.

LOCK

Indicates that the data block containing this record is to be locked when the record has been retrieved.

NOLOCK

Indicates that the data block containing this record is not to be locked following execution of this macro.

Programming Notes

The area parameter is ignored if the IOREG parameter was specified in the FCB, locate mode is assumed.

If a record is not found whose flag matches the request within the specified limits, the NOFIND address (see EXLST macro) is given control.

If LIMIT is specified or defaulted to END, file scanning proceeds to the end of the file if processing in a forward mode or to the beginning if in a reverse mode. If file search conditions are not satisfied, the EOF will be taken.

If VALTEST and LOGTEST are both specified, then both conditions must be met for the record to be retrieved.

If neither VALTEST nor LOGTEST are specified, then all records are retrieved until the limit occurs (at which a NOFIND exit is taken, if a Key limit is specified, or an EOF exit if LIMIT=END). A GETFL macro with KEYARG pointing to a logic flag mask of all binary zeros will result in a user-error exit. If processing is in the (forward, reverse) direction, LIMIT=KEY and the Key portion of the area pointed to by KEYARG is (less, greater) than the key to which the file is currently positioned, then the USERERR address of the EXLST macro will be given control.

Any attempt to direct the GETFL macro to a file created without flags, and any attempt to output a record to a flagged file which is not large enough to contain a full index (key, value flag, and logical flag) will result in the USERERR address getting control.

For flagged files, KEYARG must point to a user area large enough to contain the entire index area (key, value flag, and logical flag).

OSTAT – Receive Information Regarding Users Who Have Opened a Given File – Macro

The OSTAT macro provides the user with information about the users who have opened a given file. The information provided will be the total number of users who have the file opened currently, who have the file opened as an ISAM file with each possible OPEN/SHARUPD combination, and who have opened the file with some other access method for input or for updating.

Name	Operation	Operands
[symbol]	OSTAT	{fcb-addrx / (1)} [,{area-addrx / (0)}]

fcb

Specifies the address of the FCB associated with the file for which the OPEN status is to be obtained.

area

Specifies the area into which the information is to be placed.

Default: The macro expansion will contain a DC (Define Constant) for the area.

Programming Notes

1. Upon completion of the macro, register 1 will contain the address of the FCB and register 0 will contain the address of the area into which the information was placed. The low-order byte of register 15 will contain zero or an error code.

2. The user must have opened the file before issuing the OSTAT macro. The user's OPEN/SHARUPD combination is included in the information obtained.

3. The area into which the information is placed is described by the DIOST DSECT, which can be obtained by expanding the IDOST macro. (See "File Management DSECTS" in the appendices.)

When a task receives control at the PGLOCK exit (of the EXLST macro), its file pointers are invalid, unless the macro causing PGLOCK to be taken was a PUTX or an ELIM (no KEY), that is, the task is no longer positioned in the file. Certain ISAM macros (GET, GETR, and GETFL) require that the task be positioned before issuing the macro. The RETRY macro is provided to reposition the task in the file and, optionally, to reissue the ISAM macro which caused control to be given to the PGLOCK exit.

Name	Operation	Operand
[symbol]	RETRY	FAIL=addrx [,ACTION={POS | RETRY}] [,COUNT=integer]

FAIL

addrx specifies the location at which control is to be returned to the user if RETRY fails.

ACTION

Specifies the action to be taken by RETRY.

POS

Indicates that the task is to be repositioned in the file.

RETRY

Indicates that the task is to be repositioned in the file and the ISAM macro is to be reissued.

Default: RETRY

COUNT

Integer specifies the number of times the positioning/reissuing is to be tried before passing control to the FAIL address.

Default: 1

Programming Notes

1. This macro should only be used at the PGLOCK exit. If it is used elsewhere, a USERERR exit will be taken.

2. COUNT must be a decimal integer greater than 0 and less than 256.

3. If ACTION=POS is specified and the positioning is successful, control is returned to the task at the instruction following the RETRY macro. If ACTION=RETRY and the positioning/reissuing is successful, control is returned to the task at the instruction following the ISAM macro which was reissued. If either action fails (after being retried the number of times specified by COUNT), control is given to the task at the location specified by the FAIL parameter.

4. The task may use any register except general register 1 at the PGLOCK exit. When control is returned to the task at the instruction following the action macro as a result of RETRY reissuing the action macro, the task's registers have the same contents they would have had if the action macro had been successful originally.

5. If the last record successfully accessed prior to the PGLOCK exit is one of a sequence of records with the same key, the repositioning will position the task at the first record in the sequence.

General ISAM Programming Notes

A file may be opened in OUTIN mode even though it contains no data. The label information will be created every time a file is opened either OUTIN or OUTPUT.

ISAM does not prohibit opening of a null file (that is, a file which previously contained data but currently does not) if the OPEN type is other than INPUT. If a null file is opened INPUT, control is passed to the OPENER exit of EXLST. In the processing of a null file, an EOF exit is taken if an action macro is executed, such as a get, which depends on the existence of data records.

When creating a file with BLKSIZE=(STD,n) where n is greater than 2, the user is required to allocate n+1 PAM pages prior to opening the file. Failure to allocate sufficient space will result in an OPENER exit being taken.

Processing time required to insert new records is reduced if the user provides a good estimate of future file expansion. This estimate is specified via the PAD parameter. Although the ISAM routines attempt to split data blocks in optimum fashion when new records enter the file, such splitting requires at least one additional I/O and should be avoided.

When the action macros are being used, each of the ISAM action macros will work with an internal pointer called the "current record pointer." Generally, it is set to the record being acted upon, and modified only by the next action macro. It may point to an "imaginary location" between records.

Care should be taken in determining the file PAD value. Should the user select a PAD value which, together with the RECSIZE, exceeds the BLKSIZE, ISAM will inhibit file opening. Error code "ODBC" is used to indicate this error condition.

The current record pointer is discussed further at the end of the ISAM section.

It is very important for ISAM users to be careful when using defaulted parameters. For example, if an FCB is specified as follows:

FCB LINK=X,EXIT=Y,IOREG=9

the file will be, by default, ISAM with RECFORM=V and KEYPOS=5. These defaults are applied at assembly time (macro expansion time) and can no longer be defaulted. Therefore, if a user issues the following FILE command:

/FILE A,LINK=X,RECFORM=F,OPEN=OUTPUT

the RECFORM will indeed be changed to FIXED but KEYPOS will stay at 5.

A second example uses the same FCB and:

/FILE B,LINK=X,OPEN=OUTPUT

Since IOREG is specified in the FCB macro, the applicable ISAM macros directed to the file, once it is opened, will operate in LOCATE mode. RECSIZE is not specified, therefore RECSIZE=BLKSIZE by default. But PAD is not specified either. Therefore, 15% of the buffer will be reserved for padding. Now if a user attempts to do a PUT (in locate mode) without changing RECSIZE, RECSIZE will be greater than the area available in the buffer for user records. A user error will result.

ISAM CODING EXAMPLE

```
SOURCE STATEMENT

ISAMA     CSECT
          MCALL OPEN,PUT,CLOSE,GET,TYPIO,TERM,TERMD,EXLST,FCB
          PRINT NOGEN
          BALR  2,0
          USING *,2
          LA    6,999               *SET ORT REC COUNTER TO 999 RECS*
          OPEN  ISAMFILE,OUTPUT
MAIN1     AP    KEYP,ADDER          *ADD 1 TO KEY -- INITIALLY 0*
          UNPK  KEY+5(3),KEYP       *MOVE KEY TP RECORD*
          OI    KEY+7,X'F0'         *FORCE NUMERIC*
          PUT   ISAMFILE,RECAREA0
          BCT   6,MAIN1             *LOOP 999 TIMES*
          CLOSE ISAMFILE
          OPEN  ISAMFILE,INPUT
MAIN2     GET   ISAMFILE,RECAREA1   *RETRIEVE GENERATED RECS*
          B     MAIN2               *LOOP TILL EOF*
TRM       CLOSE ISAMFILE
          WROUT MSG,TRMD            *ISSUE OK MSG*
          LA    0,TYPE              *SET UP FOR TYPIO*
          SLL   0,8                 *SET UP FOR TYPIO*
          IC    0,X'8'              *SET UP FOR TYPIO*
          SLR   1,1                 *NO RESPONSE REQUIRED*
          TYPIO
          TERM                      *END OF PROGRAM*
TRMD      TERMD
MSG       DX    X'000D404001'

TYPE      DC    C'ISAMA OK'
KEYP      DC    P'00'
ADDER     DC    P'1'
RECAREA0  DS    0CL76
          DC    X'004C4040'
KEY       DC    8C'0'
          DC    64C'R'

RECAREA1  DS    CL76
EX        EXLST EDFADDR=TRM
ISAMFILE  FCB   RECSIZE=76,EXIT=EX
          END
```

FCB Macro Format, ISAM

Define an FCB for ISAM (0)

The following format indicates the parameters which may be supplied in the FCB macro instruction when ISAM is being used. These parameters are described in detail in "VMOS General Service File Management Macros" of the appendices.

Name	Operation	Operand
[symbol]	FCB	[LINK-linkname-symbol] [,FCBTYPE=ISAM] [,FILE=filename-name] [,PASS=password-absexp] [,RETPD=retention period-absexp] [,RECFORM= F|V,A|M|N) F|V] [,RECSIZE=absexp] [,BLKSIZE=STD|(STD,absexp)] [,IOAREA1=relexp] [,IOAREA2=relexp] [,EXIT=(relexp)|relexp] [,IOREG=reg number-absexp] [,OVERLAP=YES] [,KEYARG=relexp] [,KEYLEN=absexp] [,KEYPOS=absexp] [,PAD=absexp] [,LOGLEN=absexp] [,VALLEN=absexp] [,VALPROP=MAX|MIN] [,DUPEKY=YES] [,OPEN=INPUT|OUTPUT|EXTEND| INOUT|OUTIN] [,FORM=SHORT]

Note: The VARBLD parameter is ignored. The table (of referenced appendix) containing Summary of FCB Parameters, etc., summarizes the parameters which may be specified but are ignored depending on FCB TYPE.

PAM Macro Instruction – Read, Write, Check, or Wait a Block (S)

All user requests to DMS for a PAM action are made via this macro.

Name	Operation	Operand
[symbol]	PAM	fcbname-relexp [,RD|WRT|WT|CHK|RDWT|WRTWT|LOG] [,HP-absexp] [,LOC=1|2|relexp] [,LEN=STD|absexp]

fcbname

Specifies the address of the FCB associated with the file.

operation

Specifies the operation to be performed:

RD

Read data into memory.

WRT

Write data to the I/O device.

WT

Wait for the previous I/O to end.

CHK

Check to see if the previous I/O has terminated. If so, same as WT. If not, continue processing user's program.

RDWT

Read and wait for the read to terminate.

WRTWT

Write and wait for the write to terminate.

LOG

Requests a before-image log of the page from FRS.

Default: RDWT

HP=

Specifies the half-page number to be read or written. The first half-page of a file for PAM is page number 1. If not present, then PAM will automatically increment the half-page number by 1 for each read or write, thereby processing sequentially.

LEN=

Specifies the length of the buffer to be read or written. This value must not exceed 2048 bytes.

LOC=

Specifies typically the area to be read or written.

1

Specifies IOAREA1 address in the FCB.

2

Specifies IOAREA2 address in the FCB.

relexp

Specifies the area address in this macro.

Default: The IOAREA address in the FCB is chosen as follows:

If the last address utilized was IOAREA1, then IOAREA2 is used, and vice versa. Of course, if the IOAREA2 area does not exit, IOAREA1 is used each time.

Programming Notes

If the operation is CHK, the LOC parameter must be specified and must be in the form LOC=relexp. Control is passed to the location specified in the LOC if the "checked" operation has not yet terminated.

A DSECT, generated by the macro IDPPL, which describes the format of the PAM parameter list, is available.

For any unsuccessful entrance to PAM, control will be returned to the routine specified in the EXIT parameter of the FCB, with a code stored in the FCB. The three EXLST addresses used by PAM are:

ERRADDR = Hardware or abnormal I/O termination

USERERR = Incorrect program specification

EOFADDR = Attempt to read a dummy file

It is not necessary for the user to issue waits explicitly. PAM will automatically do a wait before any read or write. If, however, an error occurs in this wait, the return code will be 0997 instead of 0927 and the current read or write will not be executed.

When control is returned to the user after a successful read operation for which an explicit or implicit wait has been performed, the complete key is placed into the FCB. A CHK given when the I/O has completed is equivalent to a WT.

Before issuing a write operation, the program may place its own information into the last eight bytes of the key area (in the FCB) if it cares to. The first eight bytes will be constructed by PAM.

The address of the last buffer used is placed in the FCB at ID1LWB.

For the RDWT operation, the key is returned in the FCB at ID1KEY2. For all other read and write operations, ID1KEY1 is used.

PAM performs (automatic) secondary allocations if, and only if, the program attempts to write into page n+1 through n+K, where K is the number of pages specified for secondary allocation and n is the current number of pages in the file.

Note that, at OPEN time, the internal filename is placed in the leftmost four bytes of the FCB field ID1KEY1. Thus, if the program reads a page not previously written into, the internal name in that page will not match the internal filename obtained at OPEN time.

Example, PAM Coding

```
SOURCE STATEMENT

USRPAM   CSECT
         MCALL OPEN,PAM,CLOSE,WROUT,TERM,TERMD,FCB
         PRINT NOGEN
         BALR  2,0
         USING *,2
         OPEN  PAMFILE            *FCB OPEN TYPE IS OUTIN*
         MVC   BLOCK(1),='1'      *SET UP BLOCK 1*
         PAM   PAMFILE,WRTWT      *GENERATE BLOCK 1*
         MVC   BLOCK(1),='2'      *SET UP BLOCK 2*
         PAM   PAMFILE,WRTWT      *GENERATE BLOCK 2*
         MVC   BLOCK(1),='3'      *SET UP BLOCK 3*
         PAM   PAMFILE,WRTWT      *GENERATE BLOCK 3*
         CLOSE PAMFILE
         OPEN  PAMFILE,INPUT      *START SECOND PASS*
         PAM   PAMFILE,RDWT       *GET FIRST BLOCK*
         CLI   BLOCK,'1'          *CHECK FOR BLOCK 1*
         BNE   TRMD               *DUMP IF NO CHECK*
         CLOSE PAMFILE
         WROUT MSG,TRMD           *ISSUE OK MSG*
         TERM
TRMD     TERMD
MSG      DC    X'000E404001'
         DC    C'USRPAM OK'
PAMFILE  FCB   FCBTYPE=PAM,OPEN=OUTIN,IOAREA1=BLOCK,IOAREA2=NO

         DS    0F                 *WORD ALIGN FOR PAM BUFFER*
BLOCK    DS    8CL256



         END
```

FCB Macro Format, PAM

Define an FCB for PAM (0)

The following indicates the parameters which may be supplied in the FCB macro instruction when using PAM. These parameters are described in detail in "VMOS General Service File Management Macros" of the appendices.

Name	Operation	Operand
[symbol]	FCB	[LINK=linkname-symbol] ,FCBTYPE=PAM [,FILE=filename-name] [,PASS=password-absexp] [,RETPD=retention period in days-absexp] [,RECFORM=(F|V|U,A|M|N) F|V|U] [,RECSIZE=absexp] [,BLKSIZE=STD|(STD,absexp)|absexp] [,IOAREA1=NO|relexp] [,IOAREA2=NO|relexp] [,EXIT=(relexp)|relexp] [OPEN=INPUT|INOUT|OUTIN]

Note: RECFORM, RECSIZE, and BLKSIZE are accepted but are ignored by PAM action macro instructions.

SAM ACTION MACROS

FEOV – Force End of Volume (R) – Macro

The FEOV macro instruction causes the system to advance to the next (tape) volume of a file before the end of the current volume is reached.

Name	Operation	Operand
[symbol]	FEOV	{fcb-addrx (1)}

fcb

Specifies the address of the FCB associated with the file being processed.

Programming Notes

This macro is ignored for files on direct access volumes and for tape files opened REVERSE. If a successor volume does not exist, control is passed to the EOFADDR EXLST address.

GET – Get a Record (R) – Macro

The GET macro instruction retrieves the next sequential record of a file.

Name	Operation	Operand
[symbol]	GET	{fcb-addrx / (1)} [,{area-addrx / (0)}]

fcb

Specifies the address of the FCB associated with the file being processed.

area

Specifies the address of the area into which the record is moved if processing is in move mode.

Programming Notes

The area parameter is ignored if the IOREG parameter was specified in the FCB. If a record beyond the end of the file is requested, the user is given control at EOFADDR (see EXLST macro). The area parameter is ignored for files opened in update mode.

PUT – Write a Record (R) – Macro

The PUT macro instruction presents a logical record to the system for inclusion in the output file.

Name	Operation	Operand
[symbol]	PUT	{fcb-addrx / (1)} [,{area-addrx / (0)}]

fcb

Specifies the address of the FCB associated with the file being processed.

area

Specifies the address of the logical record to be moved in the output buffer if processing is in move mode.

Programming Notes

The area parameter is ignored if the IOREG parameter was specified in the FCB. After issuance of this macro in locate mode, the system places the address of the first available location in the output buffer in the register specified in IOREG. The programmer should subsequently construct the next record to be incorporated into the output file at that address.

If format-V records are processed in locate mode, the system places in the register specified by the VARBLD operand, the amount of space remaining in the output area after each PUT macro instruction is issued. It is the problem program's responsibility to ensure that the next record will fit in the available space, and to issue the RELSE macro instruction if the remaining area is insufficient.

PUTX – Replace a Record (R) – Macro

The PUTX macro instruction is used to return an updated logical record to a file. The user must not change the length of the record during the replacement process.

Name	Operation	Operand
[symbol]	PUTX	{fcb-addrx / (1)}

fcb

Specifies the address of the FCB associated with the file being processed.

Programming Notes

The PUTX macro instruction can only replace a record that was located by a locate mode GET macro instruction. The FCB must be opened UPDATE.

RELSE – Release a Buffer (R) – Macro

For input, the RELSE macro instruction causes any remaining logical records in a buffer to be bypassed. For output, the macro causes the next logical record to be written as the first record of a new block.

Name	Operation	Operand
[symbol]	RELSE	{fcb-addrx / (1)}

fcb

Specifies the address of the fcb associated with the file being processed.

Programming Notes

1. If the file is opened INPUT, REVERSE, or UPDATE, the RELSE macro instruction causes any logical records remaining in the buffer to be bypassed when the next GET is issued.

2. If the file is opened OUTPUT or EXTEND, the RELSE macro instruction causes the next logical record to be written as the first record of a new buffer when the next PUT is issued.

3. If the file is opened UPDATE and a PUTX has been issued, a RELSE will cause the entire buffer to be rewritten.

4. If the last action macro issued was FEOV or RELSE, the RELSE macro is ignored. Note that this rule overrides notes 1 and 2.

The SETL macro instruction is used to specify the position from which subsequent file processing is to take place.

Name	Operation	Operand
[symbol]	SETL	{fcb-addrx | (1)} [,position-B|E|R]

fcb

Specifies the address of the FCB associated with the file being processed.

position

B

Specifies the beginning of the file.

E

Specifies the end of the file.

R

Specifies that the file position information is to be obtained from the retrieval field in the FCB.

Programming Notes

B and E can be specified only for files on direct-access volumes or for files on single tape volumes, R can be specified only for files which have standard blocks.

For multireel (volume) tape files, the retrieval address is considered to be volume relative – not file relative.

An illegal SETL operand causes control to be sent to the USERERR EXLST address.

Example, SAM Coding

```
SOURCE STATEMENT

SAMA     CSECT
         MCALL OPEN,PUT,CLOSE,GET,TYPIO,TERM,EXLST,FCB
         PRINT NOGEN
         BALB  2,0
         USING *,2
         LA    6,999              *SET OPT REC COUNTER TO 999 RECS*
         OPEN  SAMFILE,OUTPUT
MAIN1    PUT   SAMFILE,RECAREA0   *GENERATE A RECORD*
```

```
         BCT    6,MAIN1              *LOOP 999 TIMES*
         CLOSE  SAMFILE
         OPEN   SAMFILE,INPUT
MAIN2    GET    SAMFILE,RECAREA1     *RETRIEVE GENERATED RECORDS*
         B      MAIN2                *LOOP TILL EOF*
TRM      CLOSE  SAMFILE
         WROUT  MSG,TRMD             *ISSUE OK MSG*
         LA     0,TYPE               *SET UP FOR TYPIO*
         SLL    0,8                  *SET UP FOR TYPIO*
         IC     0,=X'7'              *SET UP FOR TYPIO*
         SLR    1,1                  *NO RESPONSE REQUIRED*
         TYPIO
         TERM                        *END OF PROGRAM*
TRMS     TERMD
RECAREA0 DC     80C'R'

MSG      DC     X'000C404001'
TYPE     DC     C'SAMA OK'

RECAREA1 DS     CL80
EX       EXLST  EOFADDR=TRM
SAMFILE  FCB    FCBTYPE=SAME,RECFORM=(F,N),RECSIZE=80,EXIT=EX

         END
```

FCB Macro Format, SAM

Define an FCB for SAM (0)

The following indicates the parameters which may be supplied in the FCB macro instruction when using SAM. These parameters are described in detail in "VMOS General Service File Management Macros" of the appendices.

Name	Operation	Operand
[symbol]	FCB	FCBTYPE=SAM [,LINK=linkname-symbol] [,FILE=filename-name] [,PASS=password-absexp] [,RETPD=retention period-absexp] [,RECFORM=(F|V|U,|A|M|N) F|V|U] [,RECSIZE=absexp] [,BLKSIZE=STD|(STD,absexp)|absexp] [,IOAREA1=relexp] [,IOAREA2=NO|relexp] [,EXIT=(relexp)|relexp] [,IOREG=reg number-absexp] [,VARBLD=reg number-absexp] [,LABEL=STD|NSTD] [,OPEN=INPUT|OUTPUT|EXTEND| REVERSE|UPDATE] [,FORM=SHORT]

FCB Retrieval Address

The FCB field IDRPTR (one word) contains the retrieval address in the form

$(bbbbbbrr)_{16}$

where:

bbbbbb is the number of the buffer (not block) in the file and rr is the number of the logical record within the buffer. (For multireel tape files, the retrieval address is considered to be volume relative, not file relative.) The first record in the file has a retrieval address of 00000101.

The retrieval address values after OPEN, SETL to B and E are set as follows:

OPEN Mode	Initial Value	SETL B	SETL E
INPUT, UPDATE	00000101	00000101	bbbbbb01
OUTPUT	00000100	00000100	Error
EXTEND	bbbbbb00	00000100	Error
REVERSE	bbbbbb01	bbbbbb01	00000101

where bbbbbb is the highest buffer number plus 1 in the file.

The action macros maintain the retrieval address as follows:

GET

If the specified record causes a new buffer to be retrieved, bbbbbb is set to designate the buffer number for the record and rr is reset to 00.

PUT

If the specified record causes the existing buffer to be written, bbbbbb is set to designate the buffer number for the record, and rr is reset to 00. That is, it is set to the number of the buffer in which the record will be placed.

RELSE

If the file is opened OUTPUT or EXTEND, bbbbbb is set to the number of the buffer in which the next record will be placed; rr is set to 00.

FEOV

For tape, the retrieval address field is set to 00000100.

Programming Notes

This field is only supported for tape files which are created with standard blocks.

It is important to note that bbbbbb is buffer oriented.

Example:

Assume BLKSIZE=(STD,2)

RECFORM=F

RECSIZE=512

The retrieval address for the 10th record is 00000202. The retrieval address for the 20th record is 00000304.

Note that the system never increments the rr field beyond a single buffer. It is reset, as previously described, to zero when a new buffer is to be processed. The user can increment the rr field, if he prefers, to maintain retrieval address information for each logical record.

GENERAL

The information contained in this section defines the explicit instructions provided by VMOS for programmer control of devices at the logical level. Refer to "Macros Supporting Physical Level I/O and Run Time Parameters" in the appendices for information relating to device control.

The following summaries group the device management commands and macros into functional sets: device (space) allocation, device (space) regulation, and device (space) deallocation.

DEVICE (SPACE) ALLOCATION COMMANDS AND MACROS SUMMARY

Command/Macro	Function
FILE Command/Macro	Define a file's device requirements.
REQM Macro	Acquire a contiguous area of memory at object time.
RSTART Command	Logically attach a work station to the RBP system.
SECURE	Reserve private volume devices required for task execution.

DEVICE (SPACE) REGULATION COMMANDS AND MACROS SUMMARY

Command/Macro	Function
CHANGE Command CHNGE Macro	Alter a symbolic device name.
DROP Command	Remove the hold status of private devices. See preceding section, File Management and Access Method Definition Commands and Macros, for discussion of this command.
FILE Command/Macro	Specify the mounting of private devices.
HOLD Command	Temporarily suspend the release of private devices. See section referred to above for discussion of this command.
SYSFILE Command	Reassign SYSDTA or SYSIPT.

DEVICE (SPACE) DEALLOCATION COMMANDS AND MACROS SUMMARY

Command/Macro	Function
FILE Command/Macro	Deallocate random access space.
RELEASE Command REL Macro	Release private devices associated with a specific file. See previous section, File Management and Access Method Definition Commands and Macros, for discussion of these instructions.
RELM Macro	Release a contiguous area of user program memory at object time.
RSTOP Command	Logically detach a work station from the RBP system.

DEVICE (SPACE) MANAGEMENT COMMAND AND MACRO DESCRIPTIONS

The following material describes each of the device and space management commands and macros. The descriptions include instruction format, programming considerations, and, where illustrative information is necessary, examples of the use of the instruction. The commands and macros are presented in alphabetical order without regard to functional classification.

CHANGE Command
CHNGE Macro

The CHANGE command or CHNGE macro can be used, in the context of device management, to change the symbolic device name of a previous file definition. No other parameters introduced by the FILE command are changed.

Name	Operation	Operand
	CHANGE	[link-symbol] ,symdev-symbol

link-symbol

Specifies the file definition name.

Default: a linkname of spaces is assumed.

symdev-symbol

Specifies the VMOS symbolic device name that is to be changed.

In the context of device (space) management, the FILE instruction allocates random access space, deallocates random access space, assigns devices, and alerts the operator to mount private volumes.

Name	Operation	Operand
	FILE	[file-{filename / *DUMMY}][,LINK=symbol]
		[,DEVICE=D564|D568|D590|TA|TAPE|T9N|T9P|T7cc|WORK]
		[,MOUNT={volume sequence number-integer / (vol.seq.no,...)}]
		[,VOLUME={PRIVATE / volume serial number-alphnum / (PRIVATE,number of volumes-integer) / (volume serial number-alphnum...)}]
		[,SPACE={primary integer / (Primary integer[,secondary integer] / first page integer amount integer,ABS)}]
		[DDEVICE=D564|D568|D590]
		[,DSPACE={primary-integer / (primary-integer[,secondary-integer])}]
		[,DVOLUME={volume serial number-alphnum / (volume serial number-alphnum,...)}]

The preceding FILE command parameters are discussed in the previous section, File Management and Access Definition Commands and Macros.

RELM Macro

The RELM macro instruction releases a contiguous area of memory in a user's program at object time. Class I programs release physical memory; Class II programs release virtual memory. This macro also accepts the MF parameters.

Name	Operation	Operand
	RELM	[number={value / 1}] [,page] [,MF={L / (E,list) / (E,(1))}]

number

Specifies the number of pages to be released. If this parameter is omitted, one page is released.

page

Specifies the page number of the first page of the area to be released. This parameter must be specified by Class II programs. This parameter cannot be specified by Class I programs since memory is always released beginning with the current last page. For example when a Class I program occupying pages 1 to 10 does a RELM of 3 pages, pages 10, 9, and 8 are released. When a Class II program occupying pages 1 to 12 does a RELM of 3 pages and specifies page 10, pages 10, 11, and 12 are released.

MF=

Refer to "Command/Macro Conventions", in the appendices, for a description of this operand.

Programming Notes

1. It is permissible to release memory allocated to a program which was not obtained by the user by a previous REQM macro. For example, the program, upon loading, required five pages. After completion of the program's initialization phase, two unneeded pages can be released. (Note that loader had in fact invoked the REQM macro instruction for the user program.)

2. Memory obtained by two or more previous REQM macro instructions can be released by a single RELM, provided that these memory areas are contiguous to each other.

Cautions and Errors

General Register 15 Return Codes:

X'00'

Release processed successfully.

X'04'

Insufficient memory owned by the program to satisfy the request (for example, page parameter specifying an unallocated page or nonclass 6 memory page). The address of the first byte of the page not released is returned in the rightmost three bytes of register 1; the leftmost byte is set to 0.

X'0C'

Invalid request:

Page parameter omitted by a Class II Task, page parameter specified by a Class I task, etc.

Examples

Example 1:

Assume that the user program is a Class II program occupying pages 1-12.

Name	Operation	Operand
	RELM	3,10

Note: Pages 10, 11, and 12 are released.

Example 2:

Assume that the user program is a Class I program occupying pages 1-10.

Name	Operation	Operand
	RELM	3

Note: Pages 8, 9, and 10 are released.

Example 3:

Assume that the user program is a Class II program occupying pages 1-6.

Name	Operation	Operand
	RELM	,5

Note: Only page 5 is released.

REQM Macro

REQM requests that a contiguous area of memory be assigned to the user's program. The request is made at execution time and the memory thus assigned is taken away at program termination (or can be released by the user with the RELM macro). Memory is requested in multiples of one page. Class I programs are allocated physical memory. Class II programs are allocated virtual memory.

Name	Operation	Operand
	REQM	[n] [,first] [,MF= {L / (E,list) / (E,(1))}]

n

Specifies the number of pages requested. If this operand is omitted, the system assumes one page. A Class II program can request up to 64 pages in a single REQM. A Class I program can request up to 32 pages provided that sufficient unallocated resident, reserved, pages physically contiguous to the high order page of the program exist.

The n operand can be a general register (2-12) enclosed in parentheses. The n operand can also be an absolute expression, as defined in the Assembly Reference Manual.

first

Specifies the first page number (0-254) at which allocation is to begin. If this operand is omitted for a Class II program, the first unused area (the area with the lowest address) which can accommodate the request is allocated.

This operand is not allowed for Class I programs.

MF=

Refer to "Command/Macro Conventions", in the appendices, for a description of this operand.

Cautions and Errors

General Register 15 Return Codes:

X'00'

The request has been processed successfully. The Executive places the memory addresses of the first byte of the first page allocated in the rightmost three bytes of Register 1. The leftmost byte is set to zero.

X'04'

There is not sufficient memory available to satisfy the request. When this return code is given, no memory is allocated.

X'08'

The request is honored, but the number of pages allocated equals or exceeds the reservation limit. When this warning code is returned, the user should avoid requesting additional memory for this task.

X'0C'

An invalid request of one of the following two types:

1. A Class II program has requested more than 64 pages of memory, or a page number greater than 254.

2. A Class I program has specified the page number operand.

X'10'

The request is honored. However, the block of memory allocated crossed a segment boundary. This error is applicable only to Class II programs. This code is not returned if the page parameter (n) was specified.

Examples

Example 1:

Assume the user program is a Class I program.

Name	Operation	Operand
	REQM	

Note: One page is allocated provided it is physically contiguous to the high order page of the requesting program.

Example 2:

Assume that the program is a Class II program.

Name	Operation	Operand
	REQM	5,157

Note: Five pages beginning with virtual page number 157 are allocated, if they are available.

RSTART Command

The RSTART command logically attaches a work station to the RBP system. Once attached, the work station can monitor the RBP system for output directed to it, and users may gain access to the central system by logging on. Additionally, the RSTART command identifies the work station and defines its hardware configuration.

Name	Operation	Operand
	RSTART	termid-alphanum, termcode-alphanum

termid

Specifies the work station name. It consists of one to eight alphanumeric characters, the first of which must be alphabetic. RBP operation requires unique work station names. This command will be rejected if the system already has an attached work station with this name.

termcode

Identifies the terminal device type and defines the terminal's optional hardware features. It consists of two to eight alphanumeric characters. If omitted, the standard terminal device type for the line in use is assumed.

Programming Notes

1. The RSTART command must be the first statement submitted at an inactive work station. If the work station desires to resume RBP activity after it has been logically detached from the system (after an RSTOP command or a system failure), it must resubmit the RSTART command.

2. At present, the termcode field has no meaning. The device type is determined from the system device tables established during system generation.

RSTOP Command

The RSTOP command allows the user to logically detach a work station from the system. Prior to detachment, all queued messages directed to the work station are transmitted. No job output is returned to the work station after the RSTOP command is processed. The last message transmitted indicates that the work station is logically detached. No further communication occurs until the work station resumes RBP activity with an RSTART command.

If a work station is connected to the central system via a switched (dialed) connection, the connection is broken.

Name	Operation	Operand
	RSTOP	blank

SECURE Command

This command is used to reserve the resources that the task execution will require. The resources are devices for private volumes. These resources remain in possession of the task until the task is terminated, unless otherwise explicitly released. If a SECURE statement is received and the task already possesses one or more private device, the task will be abnormally terminated.

Name	Operation	Operand
	{SECURE / SEC}	[{T9N / TAPE}=n]
		[,T7DC=n]
		[,T7=n]
		[,D564=n]
		[,D590=n]
		[,UNIT=(mn,...)]
		[,WORK=n]
		[T9P=n]

{T9N / TAPE}=n

Indicates the number of 9-track nonphased tape devices to be secured.

T7DC=n

Indicates the number of 7-track tape devices with the data converter feature to be secured.

T7=n

Indicates the number of 7-track tape devices to be secured.

D564=n

Indicates the number of model 8564 disc devices to be secured.

D590=n

Indicates the number of model 8590 disc devices to be secured.

T9P=n

Indicates the number of 9-track phased tape devices to be secured.

UNIT=

Indicates the 2-byte installation mnemonic for the device to be secured. If more than one is used, commas separate the mnemonics. A maximum sublist of 32 items is allowed.

WORK=n

Indicates the number of work tape devices to be secured.

Programming Notes

When the WORK option is specified, the system operator must alter the required tape devices to work status.

SYSFILE Command

This command enables the programmer to reassign SYSIPT or SYSDTA.

Name	Operation	Operand
	SYSFILE	$\begin{Bmatrix}\text{SYSDTA}\\\text{SYSIPT}\end{Bmatrix} = \begin{Bmatrix}\text{mn}\\\text{CARD}\end{Bmatrix}$

mn

Specifies the 2-byte installation mnemonic of the device to which SYSIPT or SYSDTA is to be assigned.

CARD

Indicates that SYSIPT or SYSDTA is to be assigned to the card reader.

SECTION 4
COMMUNICATIONS ACCESS METHOD (CAM) COMMANDS AND MACROS

GENERAL

The information contained in this section defines the explicit instructions provided by VMOS for programmer control of its Communications Access Method (CAM). The following summary groups the commands and macros into functional sets: program and terminal identification, message processing, transmission and buffer control, and multistation line usage.

PROGRAM AND TERMINAL IDENTIFICATION COMMAND AND MACRO SUMMARY

Command/Macro	Function
CONNECT Command	Identifies a terminal to a CAM program.
PRIME Macro	Specifies the name of the program operating under CAM.

MESSAGE PROCESSING MACROS SUMMARY

Macro	Function
FETCH	Determines if an input message exists for a CAM program.
FORM	Formats a message into a specified buffer space.
LNTYP	Supplies the identification of the lines connected to a CAM program.
NOLNS	Determines the number of lines connected to a CAM program.
SEND	Transmits messages from the processor to a terminal.

TRANSMISSION AND BUFFER CONTROL MACROS SUMMARY

Macro	Function
CAMRD	Puts a read up to a specified terminal.
FBUF	Returns buffers to the buffer pool.
GBUF	Obtains buffers space from the buffer pool.
SHUTM	Inhibits further input from terminal.
WAITM	Pends a task awaiting input.
WREND	Provides the means to stop the processing activity.

MULTISTATION LINE USAGE MACRO SUMMARIES

Macro	Function
ACT	Activate a terminal.
DEACT	Deactivate a terminal.
SPSEQ	Specify polling sequence.

COMMUNICATION ACCESS METHOD COMMAND AND MACRO DESCRIPTIONS

The following material describes each of the CAM commands and macros. The descriptions include instruction format, programming considerations, and, where illustrative information is necessary, examples of the use of the instruction. The commands and macros are presented in alphabetical order without regard to functional classification.

ACT Macro (SVC $C2_{16}$)

ACT (Activate a Line or Terminal) permits deactivated lines to again be data transmission avenues, and deactivated terminals to again send or receive data.

The input required by ACT is a 3-byte field. The first byte of this field is a line number, the second byte the video data generator identification code, and the third the video data terminal identification code.

If the action is directed at a line, the second and third bytes of the byte field should contain X'00'. When the action is directed toward a terminal, the latter two bytes will vary according to the terminal type as follows.

70/751

The second byte contains the video data generator identification code. The third byte contains the terminal identification code.

70/752

The second byte contains X'00'. The third byte contains the terminal identification code.

Name	Operation	Operand
	ACT	{(1) | saddr.}

(1)

Specifies that the three input bytes are supplied in the lower order three bytes of general register 1.

saddr.

A symbolic pointer to the location of the 3-byte area.

Cautions and Errors

General Register 15 Return Codes:

X'00'

Request normally processed.

X'04'

Illegal line number of TSC (Transmission Start Code).

CAMRD Macro (SVC BC_{16})

CAMRD puts a read up to a specified terminal.

Name	Operation	Operand
	CAMRD	{addr / (1)}

addr

This entry specifies a 1-byte area containing the logical line number of the terminal.

(1)

Indicates that General Register 1 has been loaded with the line number of the terminal in the low order byte.

If the data is supplied in a parameter area pointed to by Register 1, Register 1 points to the line number.

Cautions and Errors

General Register 15 Return Codes:

X'00'

Request processed normally.

X'04'

Line number invalid.

X'08'

There is already a read on that line.

X'0C'

The specified line is processing a send.

X'10'

The specified line is disconnected.

X'14'

The specified line is deactivated.

X'18'

Unrecoverable hardware error.

X'1C'

A BREAK was received for this line.

CONNECT Command

CONNECT, a command to Attach Terminal to CAM Program, is issued conversationally by a remote terminal operator in order to attach his task to a CAM program. The conversational user must specify in the operand field of the CONNECT command a name which is defined by a PRIME macro in a CAM program. The CONNECT command is issued in lieu of a LOGON command. A terminal connected to a CAM program can transfer to the VMOS command mode by issuing a CONNECT command with VMOS as the operand.

CONNECT may also be used to attach a terminal to a different CAM program. This takes place whenever a terminal attached to a CAM program is asked by that program for additional information and this operand supplied with the CONNECT command is the name associated with another CAM program instead of VMOS. When this occurs, the terminal will be switched to come under the control of the specified CAM program. This cross-program terminal connection procedure may be accomplished at LOGON time when the Command Processor is in control, or anytime a read has been issued to the terminal when the user's program is in control.

Name	Operation	Operand
	CONNECT	{ name / VMOS }

name

Identifies the CAM program to which the terminal is to be connected. The name consists of one to eight alphanumeric characters. If a CAM program with a PRIME macro specifying the same name cannot be located, the command is rejected.

VMOS

Signifies that the issuing terminal which is already communicating with a CAM program is to be transferred to the VMOS command mode.

Caution and Errors

It should be noted that a terminal is not really connected to the program, but rather it is the telephone line number which is connected to the program. Consequently, if the terminal operator disconnects his terminal by hanging up, a subsequent dialing of the same telephone line number by anyone will result in his being automatically attached to the CAM program instead of normal VMOS mode.

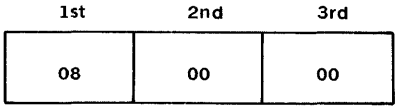
It should also be noted that the BREAK and ESCAPE functions are not supported in CAM. If either is issued from a single-station line, the terminal will be disconnected. If either is issued from a multistation line, notification of the error condition will occur when the program issues a FETCH or CAMRD. A BREAK will stop the polling and the program will have to put up a read to restart it.

DEACT Macro (SVC $C2_{16}$)

The DEACT macro (Deactivate a Line or Terminal) is used to exclude any line from having data transmitted over it or any terminal from sending or receiving data.

The input required by DEACT is a 3-byte field. The first byte of this field is a line number, the second byte the video data generator code, and the third the video data terminal identification code.

If the action is directed to a line, the second and third bytes of the field should contain X'00', e.g.,



When the action is directed toward a terminal, the latter two bytes will vary according to the terminal type as follows.

70/751

The second byte contains the video data generator identification code. The third byte contains the terminal identification code.

70/752

The second byte contains X'00'. The third byte contains the terminal identification code.

Name	Operation	Operand
	DEACT	{ (1) / symbolic addr }

(1)

The input data is in the low order three bytes of general register 1.

symbolic addr

The input data is in the three byte area pointed to by this symbolic address.

Programming Notes

Since these multistation lines are dedicated, this facility can be used to stop processing for lines or terminals that will not be used or have ceased to participate.

Caution and Errors

General Register 15 Return Codes:

X'00'

Request normally processed.

X'04'

Line number or TSC illegal.

FBUF Macro (SVC BA_{16})

FBUF (Free Buffer Space) returns buffers to the buffer pool. Before issuing the macro, the user program must load General Register 1 with the starting address of the first buffer in the string to be released. The link address in the last buffer must be 0.

A string of buffers must be released with the link information intact just as it was received from the GBUF macro.

Name	Operation	Operand
	FBUF	

Cautions and Errors

General Register 15 Return Codes:

X'00'

Request successfully processed.

X'04'

At least one of the addresses provided is not within the bounds of the buffer pool.

X'08'

User attempted to return the same buffer more than once in the same list.

X'0C'

User is trying to return a buffer he doesn't have.

X'10'

One of the returning addresses is not that of a buffer although it is an address within the buffer pool.

FETCH Macro (SVC BB_{16})

FETCH (Read Terminal Input) determines if there is an input message for the program. If a message exists, FETCH retrieves it and places it into a program-specified buffer. If no message exists, control returns to the program with an appropriate return code.

Name	Operation	Operand
	FETCH	{addr / (1)}

addr

Specifies the symbolic address of a buffer into which the incoming message is to be placed.

(1)

Indicates that General Register 1 has been loaded with the address of a buffer into which the incoming message is to be placed.

Programming Notes

1. If no message has been input for the program, the buffer is not affected. When, however, a completed input message has been received, FETCH updates the 5-byte header and places the message in the data portion of the buffer. The link field, the first four bytes of the buffer, is not affected.

2. The header contains the following information:

Bytes	Item
1-2	Count of message length including the 5-byte length of the header itself.
3-4	X'4040' for messages received from terminal on single station lines. X'40' in byte 3 and terminal TSC (Transmit Start Code) in byte 4 for 8752 on multistation line. Video Data Generator identification code in byte 3 and Terminal identification code in byte 4 for 70/751 on multistation line.
5	The logical line number of the terminal from which the message originated. The line number is assigned to the terminal by CAM when it is initially connected to the CAM program and the terminal retains the same number until it detaches from the CAM program.

Cautions and Errors

General Register 15 Return Codes:

X'00'

Normal Termination. A message for the program has been placed in the program-specified buffer.

X'04'

No input message.

X'08'

Invalid address in Register 1.

X'0C'

The message has been placed in the buffer but exceeds the buffer length. The message was truncated.

X'10'

The message has been placed in the buffer, but an unrecoverable hardware error was detected.

X'14'

The line number in Register 0 has disconnected with a read up.

X'18'

User issued a FETCH after a SHUTM and there were no outstanding reads.

X'1C'

PRIME was not issued.

X'20'

A BREAK was received from the line whose logical line number is in Register 0.

Note: Input data length is restricted to the size of one buffer, as defined in the PRIME macro.

FORM Macro (SVC $C0_{16}$)

FORM (Format a Message into Buffers for Send) takes a specified message and formats it into buffers requested from the buffer pool.

Name	Operation	Operand
	FORM	{addr / (1)}

addr

The symbolic address of the first byte of a message area. The message area must be formatted as follows:

Byte	Contents
0-1	A count which is the actual message length plus 5 for the V-field.
2-4	Reserved.
5-n	Message.

(1)

Indicates that General Register 1 has been loaded with the address of the first byte of a message area as described under addr.

Programming Notes

FORM uses the contents of byte 0-1 to calculate the number of buffers required to contain the message. FORM obtains the buffers and formats the message.

FORM returns to the user in General Register 0 the number of buffers used.

Register 1 contains the address of the first buffer.

Cautions and Errors

General Register 15 Return Codes:

X'00'

Request processed normally.

X'04'

Parameter error in GBUF.

X'08'

Buffer pool chains are destroyed.

X'10'

Insufficient number of buffers in the pool.

X'14'

Message count is invalid.

X'18'

Invalid message address.

GBUF Macro (SVC $B9_{16}$)

GBUF obtains buffer space from the buffer pool. Before issuing the GBUF (Get Buffer Space) macro, the user program must load General Register 1 with a number which indicates how many buffers are desired. GBUF will return in Register 1 the starting address of the string of buffers being provided. The first word of each buffer contains the address of the succeeding buffer. The last buffer in the string contains zeros in the link address.

Name	Operation	Operand
	GBUF	

Cautions and Errors

General Register 15 Return Codes:

X'00'

Request processed successfully.

X'04'

Parameter error.

X'08'

Buffer pool chains are destroyed.

X'10'

Insufficient number of buffers in the pool.

Note: A request for no buffers (0) is considered as a parameter error.

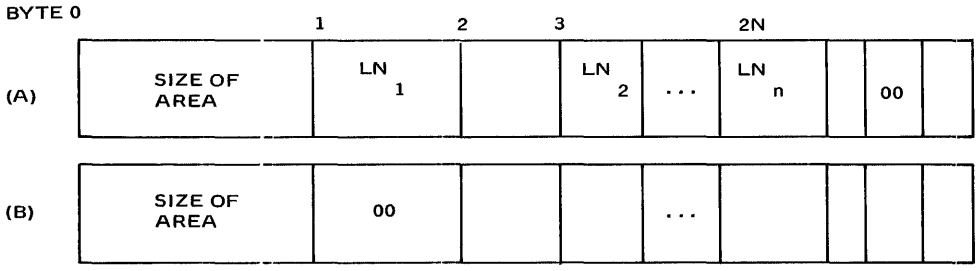
LNTYP Macro (SVC $C0_{16}$)

The LNTYP macro (List Device Types for Specified Lines) supplies the user with the device type code of any number of specific lines or a complete list of line numbers and their associated device type codes, and a count of the number of connected lines as supplied in NOLNS.

Name	Operation	Operand
	LNTYP	{addr / (1)}

addr

Specifies the symbolic address of a memory area organized according to one of the following formats.



If the area pointed to is formatted like (A), LNTYP will return control to the user program with the device type code in byte 2i+1 for the terminal or line number specified in byte 2i (i = 1,2,...n). If a request is made for a nonexistent line number, the macro will place X'FF' in the device type code byte for that line.

In the event that the area pointed to is formatted like (B), the macro will return control with a list of active line numbers in bytes 2i and their corresponding device type codes in bytes 2i+1 (i = 1,2,...n). This list will be complete provided the designated area is large enough to contain all the requested information. Should the area be too small to complete the list, control will be returned to the user when the list is full with a return code of X'08'.

The code returned for each device type supported under CAM is the same as the codes used by the TMODE macro.

01 – Model 33 TTY

02 – Model 35 TTY

03 – Model 37 TTY

04 – 8752 VDT

05 – IBM 2741

Since any of the lines may be active or inactive (that is, the user hung up but the line is still connected), the leftmost bit, 2^7, of the device type code byte is set to 1 for inactive lines and reset for active lines.

When control is returned to the user, Register 1 will contain two counts. The leftmost halfword will be the count of the total number of lines connected to the user's program, and the rightmost halfword will contain the number of those lines which are still active.

(1)

Indicates that General Register 1 contains the address of an area formatted as either (A) or (B).

Cautions and Errors

General Register 15 Return Codes:

X'00'

Request processed normally.

X'04'

There are no lines connected.

X'08'

The designated area is not large enough for a complete list of all the line numbers.

X'0C'

At least one request was made for an invalid line number.

NOLNS Macro (SVC $C0_{16}$)

NOLNS counts the number of lines still connected to the CAM program. A connected line may be either active or inactive (the user hung up).

NOLNS loads General Register 1 with the following information:

Byte	Contents
0-1	total count of connected lines.
2-3	total number of active lines.

Name	Operation	Operand
	NOLNS	

Cautions and Errors

General Register 15 Return Codes:

X'00'

Request processed normally.

X'04'

There are no lines connected.

PRIME Macro (SVC BF_{16})

PRIME is used to specify CAM program identification (the name by which the CAM program is identified). PRIME is also used to specify the size and number of buffers to be created within the buffer pool.

Name	Operation	Operand
	PRIME	name, size, number
	PRIME	(1)

name

Specifies the name by which the CAM program is identified. It may consist of from one to eight alphanumeric characters.

A terminal operator may attach his terminal to the CAM program by specifying this name in a CONNECT command.

size

Specifies the size of each buffer in the buffer pool. Each size includes a 4-byte link field, a 5-byte header, and the data field. The maximum size buffer is 2000 bytes.

number

Specifies the number of buffers to be created in the buffer pool. The maximum number of buffers is 1000.

(1)

Indicates that General Register 1 has been loaded with the address of a parameter list formatted as follows:

Byte	Item
0-1	Number of buffers
2-3	Buffer size
4-11	Name

Programming Notes

The buffer pool is created in user-accessible memory. The buffers are linked together by placing the successor buffer address into the first four bytes of the preceding buffer. The buffer chain terminates when the first four bytes are binary zeros. Each buffer string requested via the GBUF macro will be chained in the same way. The buffer pool size will not be permitted to exceed 64 pages.

Cautions and Errors

1. Only one PRIME macro can be issued in a CAM program.

2. General Register 15 Return Codes:

X'00'

Request processed successfully.

X'04'

Parameter list error.

X'08'

Name exists in CAM already.

X'0C'

PRIME issued more than once.

X'10'

Not enough memory available to create a buffer pool of the desired size.

SEND transmits to terminal a message from the processor and permits the program to request additional input from the terminal. The message length must exceed five bytes and may not be greater than 2000 bytes. However, the message may be constructed in pieces occupying more than one buffer.

Name	Operation	Operand
	SEND	{addr / (1)}

addr

Specifies the address of an Event Control Block (ECB) which contains the following information:

Byte	Contents
0	Logical line number
1-3	Address of message buffer
4	Editing options
5	Termination status
6	Special request

The editing options (byte 4) are defined in "Macro Instruction Editing Options" of the appendices.

The termination status (byte 5) contains:

X'80'

Normal termination.

X'81'

BREAK received.

X'82'

Unrecoverable hardware error on read.

X'84'

Unrecoverable hardware error.

X'88'

Read issued to deactivated line.

The special request (byte 6) is set by the program and contains:

Bit	=0	=1
2^1	After SEND is complete, do not put up a read to the line.	After SEND is complete, put a read up to this line.
2^2	Do not free the buffer(s) after transmission.	Free the buffer(s) housing the message after transmission.

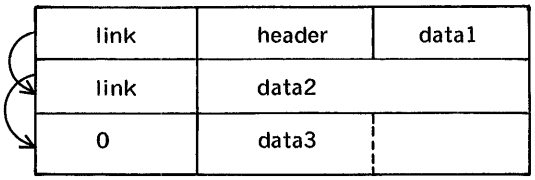
(1)

Indicates that General Register 1 has been loaded with the address of an Event Control Block as described under the addr entry.

Programming Notes

1. The ECB contains the information necessary to control message transmission. More specifically, it contains the message address, the number of the line to receive the message, edit options (refer to "Macro Instruction Editing Options" in the Appendices), a transmission status byte, and a byte for the user to select the desired options. The options available permit the user to have a read issued to the line after the write is completed and/or free the buffer containing the message at an appropriate time.

2. Messages which exceed one buffer data length appear as:



The header contains the length of the entire message, i.e., the number of bytes for the header, plus data1, plus data2, plus data3. The actual message in the last buffer need not go up to the end of buffer as is indicated by the dotted line.

3. SEND initiates the I/O and returns control to the user program before the I/O terminates. The user can now process but will not be allowed to use the same line until I/O terminates. The information pertinent to I/O termination is contained in the termination status byte.

Cautions and Errors

General Register 15 Return Codes:

X'00'

Request has been successfully initiated.

X'04'

ECB logical line number is invalid.

X'08'

The line specified is no longer connected.

X'0C'

The line is already busy either reading or writing.

X'10'

The message to be transmitted has a count greater than 2000 bytes or less than 6 bytes.

X'18'

Either the address of the ECB, the one it contains, or one of the buffer link addresses is invalid.

SHUTM Macro (SVC BE_{16})

SHUTM inhibits further input (new reads from being put up to a terminal). However, any reads outstanding when SHUTM is issued will be honored if data comes in and a FETCH is issued. SHUTM is issued when the program is preparing to terminate.

Control is returned to the program immediately following the macro.

Name	Operation	Operand
	SHUTM	

SPSEQ Macro (SVC $C1_{16}$)

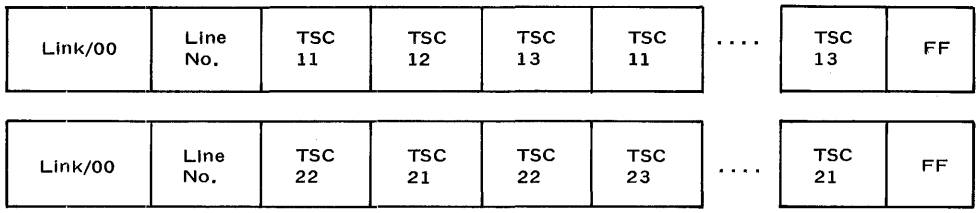
SPSEQ (Specify Polling Sequence) is provided so the user can specify the multistation lines to be connected to this program and the order in which the terminals are to be polled.

Name	Operation	Operand
	SPSEQ	{(1) | symbolic addr}

Programming Notes

The operand provides the address of a linked list specifying the desired polling sequence for each line. The address is either supplied in register 1 or symbolically as illustrated above. The lines to be connected and their polling sequences are provided through a linked list whose sublists contain a 4-byte link, the line number, the terminal TSCs for one complete polling cycle; and are terminated by a X'FF'. The last sublist is identified by a word of zeros in the link field.

The link field is four bytes long and the line number field is one byte long. The length of the TSC fields vary with the terminal type. TSC fields are one byte long. The TSC is only used when the terminal is a 70/752. If the terminal is a 70/751, the polling sequence is controlled by the hardware and the first TSC entry will be X'FF'.



After learning which lines are to be connected to this program, SPSEQ will initialize and/or create the Terminal Entry Table (TET), the Resident Terminal Entry Table (TETR), and the Remote Terminal I/O buffer for the multistation lines. However, the pointer to the poll train will not be placed in the TET until the poll train is supplied. If the line number is supplied without the poll train, CAM will generate a poll train for the line. This CAM generated poll train will poll each terminal once before restarting the cycle, thereby giving all terminals the same priority.

Additionally, SPSEQ will set up the appropriate Poll List entry for each multistation line and begin polling.

The parameter list, as well as the beginning of each sublist, must be on a word boundary.

Cautions and Errors

General Register 15 Return Codes:

X'00'

Request normally processed.

X'04'

At least one of the indicated lines is already connected to another program.

X'08'

At least one of the indicated line numbers is illegal.

X'10'

At least one of the indicated lines is not a multistation line.

X'20'

At least one TSC is incorrect.

X'40'

Parameter list or sublist not on word boundary.

Notes:

When the return code is X'04', X'08', or X'10', the affected line will be marked by an X'FF' in the first byte of the link field. Otherwise this byte will be X'00'.

When more than one of these errors occur in the same input list, the error codes will be logically added together and returned.

WAITM Macro (SVC BC_{16})

WAITM (Wait for Further Input) pends the CAM program until there is input from one of the terminals. It is intended to be used when the program cannot continue until it receives further input. Control is returned to the program immediately following the WAITM when a message has been received from a terminal.

Name	Operation	Operand
	WAITM	

WREND Macro

The WREND macro (Wait for Write Termination) provides the user with means to stop processing until the SEND he initiated out of the indicated ECB has completed.

Name	Operation	Operand
	WREND	ecbaddr [,braddr]

ecbaddr

The symbolic address of the ECB controlling the SEND.

braddr

An optional operand indicating the location to which control should be transferred after the SEND completes. If it is omitted, control will be returned to the location following the macro.

Programming Notes

SEND initiates transmission to a remote terminal and then returns control to the user, but this is no indication that the SEND is completed. When the transmission to the terminal is complete, another routine called the Write Terminator is invoked to complete the SEND. It is at this time that the completion bit is set in the ECB and the transmission is considered complete. The WREND macro stops the user's processing until such time and then returns to the user.

APPENDIX A
MACROS SUPPORTING PHYSICAL LEVEL I/O AND RUN TIME PARAMETERS

TOS MACROS

The following TOS macros, related to file processing and device management, are available to Class I programs only. The TOS FCP macros (not discussed in this document) are also available to Class I programs. Refer to TOS/TDOS FCP and Executive Communications Macros Reference Manual for the description of the TOS FCP macros, as well as additional information relating to the other TOS macros listed in this appendix.

ASSGN Macro

Dynamic Assignment (0)

Name	Operation	Operand
[symbol]	ASSGN	ccb-relexp

ccb

Specifies the address of the CCB which contains the symbolic device name of the device to be assigned.

CCB Macro

Command Control Block (0)

Name	Operation	Operand
symbol	CCB	symbolic device-symbol, ccw-relexp, flags-absexp, device type-absexp

symbolic device

Specifies the name of the symbolic device with which the CCB is associated.

ccw

Specifies the address of the first CCW to be referenced by this CCB.

flags

Specifies the user flags.

device type

Specifies the device type.

CHECK Macro

Check Completion of I/O Operation (0)

Name	Operation	Operand
[symbol]	CHECK	ccb-relexp,address-relexp

ccb

Specifies the address of the CCB to be checked.

address

Specifies the address to which control will be given if the CCB indicates that the operation has not completed.

Control will be given to the instruction following the CHECK macro if the operation has completed.

COMTY Macro

No Operation (0)

Name	Operation	Operand
[symbol]	COMTY	blank

This macro results in a no-op because it is unnecessary in VMOS to check for completion of the TYPE macro instruction. When encountered in any program, control is immediately passed to the instruction following the macro expansion. This macro is processed in this manner because messages are considered to be completed with the processing of the TYPE macro.

CPCI Macro

Check for Program Controlled Interrupt (0)

Name	Operation	Operand
[symbol]	CPCI	ccb-relexp,address-relexp

ccb

Specifies the address of the CCB.

address

Specifies the address to which control will be given if the CCB indicates that the PCI has not taken place.

Programming Notes

If the PCI has taken place, control will be given to the instruction following the CPCI.

DDEV Macro

Deallocate a Device (0)

Name	Operation	Operand
[symbol]	DDEV	sdn

sdn

Specifies the 6-byte symbolic device name of the device to be released.

Programming Notes

If the specified symbolic device name is associated with a TOS FILE command which is in the HOLD status, the device is not released. Of course, when the associated file definition is dropped, the device is released.

DMODE Macro

Obtain 7-Level Write Control Byte (0)

Name	Operation	Operand
[symbol]	DMODE	sdn,R,address-relexp

sdn

Specifies the 6-byte symbolic device name.

R

This operand is ignored.

address

Specifies the address of a 1-byte memory location into which the current write control byte for the name device will be stored.

EXCP Macro

The EXCP – Execute Channel Program (0) – performs the same function as the EXCPW except that control is returned to the user after the channel program is initiated (or queued).

Name	Operation	Operand
[symbol]	EXCP	ccb-relexp

ccb

Specifies the name of the CCB indicating the I/O operation to be initiated.

EXCPW Macro

The function of the EXCPW macro – Execute Channel Program and Wait (0) – the user's I/O channel program as specified by the CCW address in the CCB. The user is then pended until the channel program is completed.

Name	Operation	Operand
[symbol]	EXCPW	ccb-relexp

ccb

Specifies the name of the CCB indicating the I/O operation to be initiated.

QUIET Macro

The QUIET macro – Quiet Input/Output Devices (0) – waits the calling program until its I/O activity is quieted.

Name	Operation	Operand
[symbol]	QUIET	blank

SMODE Macro

Set Write Control Mode (0)

Name	Operation	Operand
[symbol]	SMODE	mode

mode

Specifies a 2-digit hexadecimal self-defining term, written in the form X'mm'. This operand specifies the write control byte which is to be stored into some PDT element.

SPRG Macro

The SPRG macro causes the purge bit in the PDT element to be set. This is reset when the device is released.

Name	Operation	Operand
[symbol]	SPRG	blank

TYPE Macro

Typewriter Requests (0)

Name	Operation	Operand
[symbol]	TYPE	message-relexp[,n-absexp] [,response-relexp] [,m-absexp]

message

Specifies the address of the message area.

n

Specifies the number of bytes in the message. If omitted, the length attribute of the message area indicates the message length. The length of a message may not exceed 127 bytes.

response

Specifies that a response to the message is required. This operand is the address of the response area.

m

Specifies the maximum number of bytes permitted in the response. If omitted, the length attribute of the response area indicates the maximum number of bytes permitted in the response. The length of a response may not exceed 72 bytes.

WAIT Macro

The WAIT macro – Wait for Completion of Channel Program (0) – is used to pend (wait) the issuing task until termination of the I/O indicated in the CCB has taken place.

Name	Operation	Operand
[symbol]	WAIT	ccb-relexp

ccb

Specifies the name of the CCB indicating the I/O operation whose completion is required.

TOS RUN-TIME PARAMETERS (RTP)

The following TOS run-time parameters are supported in the VMOS Command language. The commands are applicable to files processed by TOS FCP or physical level I/O (that is, EXCP).

FILE (ASSGN) Command (RTP)

The TOS FILE command replaces the TOS ASSGN run-time parameter. The TOS form of the FILE command is defined below and it follows TOS command syntax.

Operation	Operand
FILE	SYMDEV=symbolic device name-symbol[,LINK=symbol] [,DEVICE=D564 D568 D590 TA Tape T7cc WORK T9N T9P] [,MOUNT=mnemonic]

SYMDEV=

Specifies that a TOS FILE command is being issued. This parameter specifies the 1- to 6-character symbolic device name required by TOS.

LINK=

Specifies the linkname. Although TOS programs do not make use of this parameter, it can still be used with other command (for example, HOLD, RELEASE). For example, the RELEASE command can be used to release the private device associated with the TOS symbolic device name. If the parameter is omitted, a LINK name of "spaces" is assumed. If a FILE command has the same linkname as a previous FILE command, the new command replaces the old definition. The processing is equivalent to the sequence:

```
FILE command I      FILE LINK=X
                    .
                    .
                    .
                    RELEASE X,KEEP
                    .
                    .
FILE command II     FILE LINK=X
```

Note that any private devices associated with FILE command I are returned to the task, not the system. This important property allows the reuse of private devices. Note also that the devices associated with FILE command I are not attached (associated) with FILE command II; they are merely returned to the task.

DEVICE=

Specifies the type of device.

D564

Specifies a 8564 Disc.

D568

Specifies a 8568 Mass Storage Unit.

D590

Specifies a 8590 Disc.

TA

Specifies a 9-level or a 7-level tape with data convert. The 7-level tape is assumed to have a control code (cc) of $(F0)_{16}$; that is, odd parity, pack/unpack, and no translate.

TAPE

Specifies a 9-level tape.

T9N

Specifies a 9-level NRZ (non-return-to-zero) tape.

T9P

Specifies a 9-level phase-encoded tape.

T7cc

Specifies a 7-level tape. The control code (cc) must specify the standard write control byte required by the 7-level device (refer to table 3).

WORK

Specifies a system work tape. The system attempts to acquire a 9-level system work tape, previously specified by the operator (via the SETUP command). If no tape is available, the parameter is equivalent to the TAPE parameter.

Default: If no device type is specified, TAPE is assumed.

MOUNT=

mnemonic

Specifies the 2-character installation name of a particular device. This allows the user to specify a preference for a specified device. If the specified device is unavailable, the system attempts to acquire a device of the same type. If this fails, processing continues and a further attempt to acquire the device is made when the problem program requires it. If this parameter is specified, the DEVICE parameter is ignored unless the mnemonic specifies a 7-level tape device. In this case, the DEVICE parameter must be specified to define the control code of the tape.

Default: The system selects an appropriate device according to the device type specified in the DEVICE parameter.

FILES Command (RTP)

This command allows the user to position tape files.

Name	Operation	Operand
	FILES	sdn,n

sdn

Specifies a 1- to 6-character TOS symbolic device name.

n

Specifies the number of tape marks to be skipped (from the present position of the tape); this is a 1- to 4-digit decimal field

TPLAB Command (RTP)

This command allows the user to supply file label information for tape label checking and writing. It must immediately follow the VOL command.

Name	Operation	Operand
	TPLAB	'label'

label

Specifies a string of nonblank characters between single quotation marks.

VDC Command (RTP)

This command allows the user to specify the file identification and volume serial numbers associated with a direct access file's DTF name.

Name	Operation	Operand
	VDC	filename,matrix,identification,vsn,...

filename

Specifies the name used in the DTF macro to identify the file.

matrix

Specifies the number of bytes to be reserved for the extent matrix for this file. This parameter is 1- to 4-digit decimal field. When created, this area is preceded by a 2-byte field which contains the length in binary of this area. If this parameter is omitted, this 2-byte field is set to zeros.

identification

Specifies the name in the file label.

vsn

Specifies the volume serial number or numbers associated with this file. Each vsn is a 6-character alphanumeric field.

VOL Command (RTP)

This command allows the user to associate a TOS symbolic device name (sdn) with a filename for checking or writing standard labels for tape files. If this command follows a FILES command and the sdn's do not agree, the command is rejected.

Name	Operation	Operand
	VOL	sdn,filename

sdn

Specifies 1- to 6-character TOS symbolic device name.

filename

Specifies the filename associated with the DTF.

APPENDIX B
MISCELLANEOUS TASK AND FILE (DATA) MANAGEMENT COMMANDS AND MACROS (CLASS I PROGRAMS)

GENERAL

The commands and macros listed in this appendix define operations available to the user in the areas of Task and File Management. They have been segregated from the general discussions of these areas as, although they provide useful features, they are not requisite to Task and File Management.

VMOS TASK MANAGEMENT COMMANDS AND MACROS

The following commands and macros may be used with Class I and Class II programs.

ACCNT Macro	Allows the user to add an identifier to his accounting macros.
ACCOUNT Command	Allows user or System Administrator to manage accounting files.
ASCII Macro	Allows the user to set the decimal code for Processor state 1 to ASCII.
BCNTRL Command	Control broadcast and message transmission.
EBCD Macro	Allows the user to set the decimal code for Processor state 1 to EBCDIC.
GDATE Macro	Allows the user to place the current date in his program.
GEPRT Macro	Allows the user to obtain the elapsed CPU time for his task.
GETOD Macro	Allows the user to insert the time of day into his program.
GTMAP Macro	Allows the user to obtain a bit map of a task's virtual memory.
REMARK Command	Allows the user to output remarks to the SYSOUT file.
SETBF Macro	Enables the user to change the size of the terminal buffer.
TYPE Command	Allows the user to send a message to the operator's console.

ACCNT Macro

The ACCNT macro permits the user or administrator to add an 8-character identifier to his accounting records.

Name	Operation	Operand
	ACCNT	{ id-addr (r) }

id-addr

Specifies the address of an 8-byte identification area.

(r)

Specifies the general register containing the address of an 8-byte identification area.

Cautions and Errors

General Register 15 Error Return Codes

X'00'

Addition was successful.

X'04'

Invalid address supplied.

ACCOUNT Command

The ACCOUNT command enables the user and/or system administrator to manage files of records containing information concerning task processing. Among those management options provided by this command are renaming and redirecting accounting files, monitoring CPU time for accounts, communicating to the operator the names of user initiated programs, JOIN File interrogation, and the insertion of identifiers into accounting records.

Name	Operation	Operand
	ACCOUNT	[CHANGE=filename],
		[RECORD={ALL | userid | (userid,...)}]
		[ID={C'text' | X'text'}]

CHANGE=filename

User or System Administrator: specifies the new accounting file to be opened. The current file will be automatically closed. A maximum of 54 characters is allowed.

RECORD=ALL

System Administrator Only: requests the spooling out of a list of all userids currently active in the JOIN File and such related information as mailing addresses, DMS page allocation values, account number, maximum priorities, and CPU time values.

RECORD=userid

User: Displays accounting information and password associated with the designated userid.

RECORD=(userid,...)

System Administrator: lists the accounting information and password associated with the designated userid.

RECORD=(userid,...)

System Administrator Only: lists all accounting information and passwords for those userids listed.

ID=C'text'

User or System Administrator: inserts one to eight alphanumeric characters expressed as a character constant into all the user's accounting records.

ID=X'text'

User or System Administrator: inserts one to eight alphanumeric characters, expressed as a hexadecimal constant, into all the user's accounting records.

Programming Notes

Only one operand may be specified in a given issuance of the ACCOUNT command. To specify a new accounting file, display the user's ID, and insert an identifying string into the user's accounting records by issuing the following three commands.

```
ACCOUNT CHANGE=TJFILE
ACCOUNT ,RECORD=$TJREC
ACCOUNT ,,ID-X'1234'
```

The combination of any or all of these in one command statement is not permitted.

ASCII Macro

The ASCII macro permits the user or administrator to set the ASCII mode. Actually, ASCII is used to set the processor decimal code for Processor State 1 to ASCII for the task in which the program is running.

Name	Operation	Operand
	ASCII	

Programming Notes

The execution of this macro causes the ASCII decimal code to be used in determining the zone configuration when executing the decimal arithmetic instruction set, the Edit instruction, and the Edit and Mark instruction.

The setting of this decimal code does not affect any automatic translation of data read into or written from the processor.

BCNTRL Command

Broadcasts and messages can be allowed or prohibited at the discretion of the user. However, emergency messages as determined by the system will be sent to SYSOUT; e.g., all shutdown messages.

Name	Operation	Operands
	{BCNTRL / BC}	[MES = {Y / N}] [,BCST = {Y / N}]

MES =

Y

All text sent by the /MESSAGE command will be received on SYSOUT.

N

All text sent by the /MESSAGE command will not be received on SYSOUT.

BCST =

Y

All text sent by the /BROADCAST command will be received on SYSOUT.

N

All text sent by the /BROADCAST command will not be received on SYSOUT.

Programming Notes

Prior to the first use of this command, all text sent by the MESSAGE and BROADCAST commands will be received.

EBCD Macro

EBCD enables the user to set the EBCDIC mode. Actually, it is the setting of the processor decimal code for Processor State 1 to EBCDIC for the task in which the program is running.

Name	Operation	Operand
	EBCD	

Programming Notes

The execution of this macro causes the EBCDIC decimal code to be used in determining the zone configuration when executing the decimal arithmetic instruction set, the EDIT instruction, and the Edit and Mark instruction.

The setting of this decimal code does not affect any automatic translation of data read into or written from the processor.

GDATE Macro

GDATE acquires the current date and places it in a location supplied by the user.

Name	Operation	Operand
	GDATE	{location / (1)}

location

Specifies the symbolic name of the 12-byte area in the user's program into which the Executive is to place the current date.

The date is stored in zoned decimal format as mm/dd/yyjjjb where:

mm

month (two bytes),

dd

day of month (two bytes),

yy

year (two bytes),

jjj

day of the year (Julian date), (three bytes), and

b

blank character (X'40'), (one byte).

Note: The Executive stores slashes (/) in memory just as shown in the above format.

(1)

General Register 1 which has been loaded by the user with the address of the field to receive the current date.

Examples

Example 1:

A sample date is shown below (in hexadecimal notation) as it would appear in the user-supplied area following the execution of GDATE.

Assume the current date to be 29 February 1968.

BYTE	0	1	2	3	4	5	6	7	8	9	10	11
HEX	F0	F2	61	F2	F9	61	F6	F8	F0	F6	F0	40
CODE	m	m	/	d	d	/	y	y	j	j	j	b
DECIMAL	0	2	/	2	9	/	6	8	0	6	0	
	MONTH			DAY			YEAR		JULIAN DAY			BLANK

Example 2:

Name	Operation	Operand	
	LA	1,AREA	①
	GDATE	(1)	②
	.		
	.		
	.		
AREA	DS	CL12	③

① The user loads General Register 1 with the address of the field to receive the current date.

② The GDATE macro is issued referencing General Register 1.

③ AREA is a 12-byte field reserved for receipt of the current date.

GEPRT Macro

GEPRT (Get Program Time) acquires the task's elapsed CPU time.

Name	Operation	Operand
	GEPRT	location

location

The symbolic name of the leftmost byte of a 6-byte area in the user's program into which the amount of elapsed CPU time currently used by this task is stored.

CPU time is stored in zoned decimal format as hhmmss where:

hh

hours (two bytes),

mm

minutes (two bytes), and

ss

seconds (two bytes).

Examples

Example 1:

08 27 00 (8 hours and 27 minutes) would be stored as:

F0F8F2F7F0F0

Example 2:

Name	Operation	Operand
	GEPRT	TIME
	.	
	.	
	.	
	.	
TIME	DS	CL6

The task's elapsed CPU time is stored in the user's program at location TIME.

GETOD Macro

GETOD (Get Time of Day) causes the Executive to store the current time of day in a location specified by the user.

Name	Operation	Operand
	GETOD	{location / (1)}

location

Specifies the symbolic address of the location into which the Executive is to store the current time of day.

The time is stored in zoned decimal format as hhmmss where:

hh

hours (two bytes),

mm

minutes (two bytes), and

ss

seconds (two bytes).

For example: 13:09:27PM (27 seconds after 9 past 1 PM) would appear as:

hh mm ss

F1F3F0F9F2F7

(1)

General Register 1 which has been loaded by the user with the address of the field to receive the current time of day.

GTMAP Macro

GTMAP (Get Memory Map) is used to obtain a bit map of a task's virtual memory and the number of the task's last virtual memory page.

The user must allocate 34 contiguous bytes for receipt of the information. The first 32 bytes contain the bit map of the user's virtual memory. Each bit represents a virtual page. A bit setting of 1 indicates an unallocated (free) page.

The last two bytes of the 34-byte area will contain the number of the last virtual memory page.

For example: if the last page is 239_{10}, the last two bytes of the GTMAP area would be $00EF_{16}$.

Name	Operation	Operand
	GTMAP	{location / (1)}

location

Specifies the symbolic address of a 34-byte area into which the Executive is to store the user's virtual memory bit map.

(1)

General Register 1 which has been loaded by the user with the symbolic address of the field to receive the virtual memory map.

Cautions and Errors

General Register 15 Return Codes:

X'00'

Request processed successfully.

X'04'

The user's receiving field address is not within user's memory.

X'08'

User Virtual Memory Table has not been allocated.

REMARK Command

REMARK allows the user to output remarks to the SYSOUT file. For a conversational user who is not in the PROC mode, remarks are not written.

Remarks cannot be contained on multiple lines, but any number of REMARK commands can be issued.

Name	Operation	Operand
	REMARK	remarks

remarks

Any desired remarks can be entered. A remark should not exceed 71 characters in length.

Programming Notes

If the REMARK command is used in a procedure file, the command print option in the PROCEDURE command must be exercised for the remark to be written to the SYSOUT file.

Examples

Example 1:

```
/REMARK THIS GOES TO SYSOUT
```

The remark "THIS GOES TO SYSOUT" is output to the SYSOUT file. If SYSOUT is the user's terminal, the remark is not written.

Example 2:

```
/.NAME REMARK REMARKS
```

This command has a name field. The remark "REMARKS" is output to the SYSOUT file.

SETBF Macro

The SETBF (Set Buffer) macro instruction is issued in a user's program to change the size of the terminal buffer. The buffer size may range from 80 to 1024 bytes.

Name	Operation	Operand
	SETBF	{size[,N] / (1)}

size

An integer between 80 and 1024 which defines the size of the terminal buffer desired.

N

Indicates no change is to be made in the buffer size if the requested size (size defined above) is at least as large as the current buffer size.

(1)

Indicates the user has loaded General Register 1 with the size of the buffer desired before issuing the SETBF macro.

Programming Notes

If the program using the SETBF macro is in the nonconversational mode, no action will be taken because the SYSIN and SYSOUT files will handle a record up to 1020 bytes.

Error Codes

General Register 15 Return Codes

X'00'

Requested buffer obtained.

X'04'

Invalid buffer size provided.

X'08'

New buffer could not be obtained.

TYPE Command

TYPE is issued by the user to cause a message to be printed on the system operator's console typewriter.

TYPE can be issued in either the conversational or nonconversational mode.

Name	Operation	Operand
	TYPE	msg

msg

Specifies the message to be typed on the operator's console typewriter. This message cannot exceed 72 characters in length. Any characters that are on the typewriter keyboard are acceptable.

Examples

```
/.NAME TYPE MESSAGE=HELLO, JOE     ①

/TYPE OK CHARACTERS ARE A-Z,1-9,.#@'='^ →(%;-+|!$-C)*,.AND/.     ②

/TYPE 72 CHARACTERS MAX. "72 CHAR. MAX."     ③
```

① This named command causes the message "MESSAGE = HELLO, JOE" to be typed on the operator's console typewriter.

② This command does not have a continuation line. The 58 character message shows all the characters acceptable in a TYPE command. These are the letters A-Z, the digits 0-9, and the special characters as shown.

③ The message "72 CHARACTERS MAX." is typed on the operators console. This command has a comments field, which is not typed on the operators console.

The following macros are restricted to use with Class I programs.

ADEXT Macro	Allows a user's Class I program to obtain the address of the Executive Communication Region (ECR) and Program Table (PT) entries for the user program.
FLOAD Macro	Allows a user's Class I program to load in FCP overlay.
MONTB Macro	Allows the user to get the address of the monitor table in order for his program to interrogate the elements of the table.

ADEXT Macro

ADEXT macro (Address Executive Tables) is used in a Class I program to obtain the address of the Executive Communication Region (ECR) and Program Table (PT) entries for the user program. Portions of the Tape Operating System ECR and PT are simulated in the VMOS to provide intersystem compatibility. For description of the ECR and PT, refer to "VMOS Support of TOS Tables" in the appendices.

Name	Operation	Operand
	ADEXT	

Programming Notes

After the macro is executed, the user program receives control at the instruction following the macro expansion.

General Registers 14 and 15 contain the following:

Register 14

Address of the first byte of the Executive Communication Region.

Register 15

Address of the first byte of the user program's Program Table Entry.

The address in bytes 24-27 of the Program Table can be used to obtain information contained in the program's Executive Storage Area.

Cautions and Errors

ADEXT is ignored if it appears in a Class II program.

FLOAD Macro

FLOAD macro (Load FCP) enables a Class I program to load an FCP overlay.

Name	Operation	Operand
	FLOAD	symbol,laddr[,raddr]

symbol

Specifies the name of the FCP overlay to be loaded. The symbol may be one to six characters in length.

laddr

Specifies the load address.

raddr

Specifies the address to return to alter the module has been loaded. If this operand is omitted, control is returned to the instruction following the macro expansion.

MONTB Macro

MONTB macro (Address Monitor Table) enables a system program to get the address of the Monitor table in order for the program to interrogate the elements of the table. The macro causes the VMOS Executive to provide the user program with the address of the Monitor table. This address is in class 6 memory in general register 15. Control is returned to the user.

Name	Operation	Operand
	MONTB	blank

blank

Indicates the absence of any operand designation.

TOS FILE MANAGEMENT MACROS

The following macros may be used with either Class I or Class II programs.

EXLST Macro	Enables the user to provide the operating system with an address to which control may be given for a contingency situation.
EXRTN Macro	Enables the user to provide the operating system with information upon return from EXLST processing.
TYPIO Macro	Enables the user to have his program send and receive information to and from the operators console.

During the opening of a file or while processing is occurring, a situation may arise which should be reported to the program. Such situations, Contingency and Error Exits, may arise from an error on the part of the program or they may be external to the program. The program is also notified when such things as label processing must occur. The EXLST macro instruction is a means whereby the program provides DMS with an address (or addresses) where control may be given when such a situation arises. The program may use three techniques to inform DMS of its needs:

1. Provide the address of one routine to process all conditions.

2. Provide the addresses of routines to process any conditions relevant to the program. Note that some conditions will result in the program's termination if no processing routine is provided.

3. Provide the address (or addresses) of routine(s) to process a particular condition as well as the address of a routine to handle all possible contingencies.

No matter which technique is used, the following conventions obtain:

1. The address of the file's FCB is placed into register 1 before the program is given control.

2. The address of the instruction where control would have passed is placed into the FCB.

3. Each possible condition has an identifying number which will be stored in the FCB before the program receives control.

If the program uses one routine to process all contingencies, its address should be placed into the FCB. Otherwise, the address of an EXLST macro instruction is placed into the FCB.

The EXLST Macro Instruction

Name	Operation	Operand
[SYMBOL]	EXLST	[OPENX=relexp|NO] [,OPENER=relexp|NO] [,PASSER=relexp|NO] [,LOCK=relexp|NO] [,NODEV=relexp|NO] [OPENZ=relexp|NO] [,LABGN=relexp|NO] [,LABEOV=relexp|NO] [,CLOSER=relexp|NO] [.OPENER=relexp] [,LABEND=relexp|NO] [,EOVCTRL=relexp|NO] [,OPENV=relexp|NO] [,EOFADDR=relexp|NO] [,ERRADDR=relexp|NO] [,ERROPT=name|IGNORE|SKIP] [,WLRERR=name|NO] [,NOSPACE=relexp|NO] [,ISPERR=relexp|NO] [,DUPEKY=relexp|NO] [,NOFIND=relexp|NO] [,USERERR=relexp|NO] [,SEQCHK=relexp|NO] [RECON=relexp|NO] [,COMMON=relexp] [,PGLOCK=relexp]

In all cases, the relexp operand is the address of a routine to handle the associated condition. If the NO option is taken, the program is indicating that it does not wish control upon occurrence of the corresponding event. The COMMON exit allows the program to provide one routine to handle all conditions that do not require a tailored routine. In all cases where NO is an option, this is also the default value. To nullify the default, it is necessary to specify the keyword with a null operand.

A discussion of the LABGN, LABEOV, and the LABEND operands will be found under "Tape Labels and Tape Label Processing", in the appendices of the VMOS Programmer's Reference Manual, since these parameters are used for the processing of either user labels or nonstandard labels.

OPENX=

The FCB has been modified by the information supplied either in a FILE command (macro) or through the catalog. The program may now ensure that all parameters are consistent so that the OPEN can be completed with no errors. If a new file is being created, then the FCB has been modified by the FILE command, but the catalog entry has not yet been written.

OPENER=

An error was encountered while attempting to OPEN the file (for example, inconsistent FCB, no space allocated for the file, or a null ISAM file opened INPUT). An error code detailing the condition is stored in the FCB.

PASSER=

An invalid password was specified for a protected file.

LOCK=

For a non-ISAM file, the condition was encountered because the program attempted to open a file in a mode other than INPUT and it was already open by another user. Alternatively, if the file is open currently by another user in a mode other than INPUT and the program tries to open the file INPUT, the same condition occurs. For a non-ISAM file, the LOCK exit indicates that the OPEN/SHARUPD specifications conflict with the OPEN/SHARUPD specifications of other users who have already opened the file.

NODEV=

No free device exists upon which the private volume may be mounted; or, the private volume is presently being used by another user. If the SECURE, HOLD, and DROP commands are used properly, this condition should not arise.

OPENZ=

For a file being opened either OUTPUT or OUTIN, the catalog processing has been completed by the time the exit is taken; however, the remainder of the open process remains to be done. The program may modify the FCB at this time so that processing is done in a way convenient to it. For example, the catalog might indicate that a file was SAM; however, the user might want to process it using PAM.

CLOSER=

An error was encountered while attempting to CLOSE the file (for example, error encountered when attempting to write labels). An error code detailing the condition is stored in the FCB.

PGLOCK=

This parameter is applicable only if the file was opened with SHARUPD=YES. Control will be given to the user at this exit whenever a data block (or index page) which must be referenced in the course of processing an action macro is inaccessable because of the manner in which another user is accessing it. (For example, if one user has locked a data block and a second user tries to lock it, the second user will be given control at his PGLOCK exit.)

It is possible for any macro issued for a file opened with SHARUPD=YES to cause control to be passed to this exit. Unless caused by a PUTX or an ELIM (no KEY), the "internal pointer" will be invalid when the PGLOCK exit is taken. Thus, it is necessary to reposition this pointer before issuing a macro which assumes that it is valid (such as GET, GETR, and GETFL). The pointer can be repositioned by issuing the RETRY macro or one of the following ISAM action macros: GETKY, SETL, PUT, STORE, INSRT, or ELIM (KEY). If a GET, GETR, or GETFL is issued before the pointer is repositioned, control will be passed to the USERERR exit.

If the macro which caused the PGLOCK exit to be taken was a PUTX or an ELIM (no KEY), the data block will still be locked at the PGLOCK exit and no repositioning is required.

EOVCTRL=

Label processing is completed after a new volume is mounted. The FCB will contain the 6-digit Volume Serial Number (VSN) of the new volume. Initially, the VSN of the first volume will be stored in the FCB. This exit is only activated if the file is on multiple reels.

OPENV=

This exit is taken when a tape volume with nonstandard labels is encountered.

EOFADDR=

The end of file condition has occurred when a read was attempted.

ERRADDR=

A hardware malfunction or an abnormal I/O termination condition has occurred during the processing of the file. Status bytes consisting of the standard device byte, the executive flag byte, and the three sense bytes are stored into the FCB. Note that SAM read errors use other exits.

ERROPT=

This entry applies to SAM direct access or tape input files, and it specifies functions to be performed for an error block.

If a parity error is detected when a block of records is read, the block is reread a standard number of times before the block is considered an error block. After this, the job is automatically terminated, unless the ERROPT entry is included to specify other procedures to be followed on an error condition. Either IGNORE, SKIP or the symbolic name of an error routine can be specified. The functions of these three specifications are:

IGNORE

The error condition is completely ignored, and the records are made available to the user for processing.

SKIP

No records in the error block are made avaible for processing. The next block is read from the device and processing continues with the first record of that block.

name

DMS branches to the user's routine where he can perform whatever functions he desires, to process or make note of the error condition. Register 0 contains the address of the block in error.

In his error routine, the programmer must not issue any GET instructions for the records in the error block. In this routine, the user may issue logical (GET, PUT) macros to any file other than one in error. If the program wishes to return to normal processing, the EXRTN macro must be issued. If the register 0 contains X'00000001', then the current block will be skipped, and processing continues with the next block. Any other code indicates that this block is to be processed as if no error occurred.

This entry does not apply for output files. The entry applies to wrong-length records if the entry WRLERR-name is not included.

WLRERR=

A wrong-length record was read from a tape file. Whenever fixed-length blocked records or variable-length records are specified, the machine check for wrong-length records is suppressed, and DMS generates a program check of record length. For fixed-length blocked records, record length is considered incorrect if the physical tape record (gap-to-gap) that is read is not a multiple of the logical record length (specified in FCB entry RECSIZE), up to the maximum length of the block (specified in FCB entry BLKSIZE). This permits the reading of short blocks of logical records without a wrong-length record indication. For variable-length records, record length is incorrect if the length of the tape or direct-access record is not the same as the block length specified in the block count control field.

When the program is given control, register 0 contains the address of the block in error. The EXRTN macro may be issued to continue processing. If register 0 contains X'00000001', then the current block will be skipped and processing continues using the next block. Any other code indicates that this block is to be processed as if no error had occurred.

If the WLRERR entry is omitted and a wrong-length record is detected by DMS, one of the following occurs:

1. If the FCB entry ERROPT is specified for the file, the wrong-length record is treated as an error block and processed according to the program's specifications for an error.

2. If the FCB entries ERROPT and ERRADDR are not specified, the job is terminated.

The WLRERR entry does not apply to undefined records. Undefined records are not checked for incorrect record length.

NOSPACE=

Insufficient space exists with which to perform secondary allocation or extended output. If preassigned tape volumes exist, then all volumes have been filled.

ISPERR=

Insufficient space exists to expand the index of an ISAM file. Note that an ISAM file may have to be extended to obtain room either for data or index although it is one file.

DUPEKY=

A matching key exists in the file and the INSRT action macro was issued.

NOFIND=

A record with the matching key could not be found when the GETKY macro was issued or a record which matched the GETFL criteria could not be found within the specified range.

The user will be given control at this exit if the file is opened with SHARUPD=YES and he issues a PUTX or ELIM (no KEY) without first locking the data block or if he issues a GET, GETR, or GETFL after taking his PGLOCK exit without first repositioning his file.

USERERR=

The program has tried to perform an illogical use of the action macros or has issued an illegal action. Examples are issuing a write macro to a file opened input or an improper operation code in PAM.

SEQCHK=

A record being added to a file via the PUT macro in ISAM has a key less than the highest key already present in the file or a PUTX attempted to change the key.

The user will be given control at this exit if the file is opened with SHARUPD=YES and he issues a PUTX or ELIM (no KEY) without first locking the data block or if he issues a GET, GETR, or GETFL after taking his PGLOCK exit without first repositioning his file.

COMMON=

Specifies that all exit conditions, except those noted below, are to exit to the location specified by COMMON. If any other conditions (that is, other parameters) are specified, they are routed to their specified locations. In other words, COMMON allows programming by exception.

Note that COMMON does not apply to the following label and/or OPEN contingency exits:

OPENX	OPENZ	OPENV	LABGN
LABEOV	LABEND	EOVCTRL	ERROPT
WLRERR			

EXRTN Macro

Name	Operation	Operand
[symbol]	EXRTN	{fcb-addrx / (1)} {,disposition-value / (0)}

fcb

Specifies the address of the FCB of the file whose processing resulted in the contingency exit.

disposition

0

Label verification (OPENV) was satisfactory. Continue OPEN process.

1

Error discovered in label verification (OPENV). The OPEN process will notify the operator.

Table B-1 summarizes the use of the contingency exits.

TYPIO Macro

The TYPIO macro (Available to Class I and Class II Programs) (0) performs the same function as the TOS TYPE macro instruction. However, it facilitates the writing of reentrant programs because the operands are passed in registers.

Name	Operation	Operand
[symbol]	TYPIO	blank

Programming Notes

Parameters are passed in registers. The leftmost three bytes of register 0 contain the address of the message to be output. The rightmost byte contains the message length.

If a response to the message is not required, register 1 must contain zeros. Otherwise, the leftmost three bytes of register 1 contain the address of the area to receive the response; the rightmost byte contains the area size, which thus specifies the maximum message size. Longer messages are truncated without error notification.

If the input message is less than the input area size, the message is left justified in the input area. In this case, a byte of binary zeros is appended to the message.

TABLE B-1. SUMMARY OF THE USE OF THE CONTINGENCY EXITS

Contingency Exits	STD SAM	NSTD SAM	User May Issue						FCB Code	Terminate (1)	Comments
			ISAM	PAM	BTAM	CLOSE	EXTRN or LBRET	ACTION Macro			
OPENX	A	A	A	A	A	A	A	N	4	N	
OPENZ	A	A	A	A	A	A	A	N	24	N	
OPENV	A	A	N	N	A	A	A		28	N	
OPENER	A	A	A	A	A	N	N	N	8	A	Error code stored in FCB.
PASSER	A	A	A	A	A	N	N	N	12	A	
LOCK	A	A	A	A	A	N	N	N	16	A	
NODEV	A	A	A	A	A	N	N	N	20	A	
LABGN	A	A	N	A	A	X	A	A	36	N	
LABEOV	A	A	N	A	A	X	A	A	40	N	
CLOSER	A	A	A	A	A	N	N	N	44	N	File considered closed.
LABEND	A	A	N	A	A	X	A	A	48	N	File considered closed.
EOVCTRL	A	A	N	N	A	A	A	N	52	N	
EOFADDR	A	A	A	A	A	A	N	A	64	A	

(Continued)

Legend:

A = Allowed
X = Not Allowed
N = Not Applicable
(1)= Program is terminated if exit not specified and action occurs.

TABLE B-1. SUMMARY OF THE USE OF THE CONTINGENCY EXITS (Continued)

Contingency Exits	STD SAM	NSTD SAM	User May Issue						FCB Code	Terminate (1)	Comments
			ISAM	PAM	BTAM	CLOSE	EXTRN or LBRET	ACTION Macro			
ERRADDR	A	A	A	A	A	A	N	A	68	A	Error Code and/or termination bytes (PAM and BTAM).
ERROPT	A	A	N	N	N	A	A	N	72	A	Termination bytes stored in FCB.
NOSPACE	A	N	A	N	N	A	N	N for SAM; allowed otherwise.	76	A	
ISPERR	N	N	A	N	N	A	N	A	80	A	
DUPEKY	N	N	A	N	N	A	N	A	84	A	
NOFIND	N	N	A	N	N	A	N	A	88	A	
USERERR	A	A	A	A	A	A	N	A	92	A	Error code stored in FCB.
SEQCHK	N	N	A	N	N	A	N	A	96	A	
WLRERR	A	A	N	N	N	A	A	N	100	N	Termination bytes stored in FCB.

Legend:

A = Allowed
X = Not Allowed
N = Not Applicable
(1)= Program is terminated if exit not specified and action occurs.

APPENDIX C
MACRO INSTRUCTION EDITING OPTIONS

The following table lists the editing options available to the problem program. These editing options are performed by the Remote Terminal Input/Output (RTIO) routines on messages to or from a remote terminal.

Edit Option	Writes to Terminal	Reads from Terminal
$2^0 = 0$	Translate output from EBCDIC to appropriate output code.	Translate input to EBCDIC.
$2^0 = 1$	No translation.	No translation.
$2^1 = 0$	Insert line feed and carriage return characters at the beginning of text and every 72 character thereafter.	Delete line feed and carriage return characters.
$2^1 = 1$	No insertion of line feed and carriage return characters.	No deletion of line feed and carriage return characters.
$2^2 = 0$		Performs the backspace operations.
$2^2 = 1$		Retain the backspace characters and do not perform the backspace operation.
$2^3 = 0$		Continue reading from the normal source, i.e., if paper tape was the previous input source, read next record from paper tape.
$2^3 = 1$		Reset from paper tape option to manual (keyboard) control. This facility is to be used with paper tape so errors can be corrected on the keyboard.
$2^4 = 0$	Messages are always outputted.	Translate all lower case alphabetic characters to upper case EBCDIC alphabetics.

Edit Option	Writes to Terminal	Reads from Terminal
$2^4 = 1$	If input is prepared (for example, paper tape) do not output this message.	True translation, i.e., lower case alphabetics to lower case EBCDIC alphabetics.
$2^5 = 0$	Do not append print control characters to message.	
$2^5 = 1$	Put paper tape control characters preceding message. (This option is used to prepare a paper tape for subsequent reading by VMOS.)	
$2^6 = 0$	Do not scan message for new line character.	
$2^6 = 1$	For 2741 only – scan message for new line character and insert the required number of idle characters. If the number of characters between two new line characters plus the number of required idle characters exceeds the 2741 type bar length, the message is truncated. Also, if the total number of characters comprising the expanded message exceeds the buffer size, the message is truncated.	

APPENDIX D
VMOS SUPPORT OF TOS TABLES

GENERAL

The VMOS Executive is supporting TOS programs by supplying the Executive Communications Region (ECR), Program Table (PT), the Executive Storage Area (ESA), and the Monitor Table. The ECR, PT, and ESA are created in physical core adjacent to the Class I program at RUN time. The area occupied by these blocks will hereafter be referred to as the TOS Communication Area (TCA). The Job Control Block (JCB) has a pointer to the TCA which is initialized to zero when the TCA has not been created. When a Class I program is loaded, the program size parameter is incremented to include the displacement of the TCA. Core is obtained, the address of the TCA is placed in the TCA pointer in the JCB, and the blocks are formatted. The TCA is released at job termination. Each Class I program has a TCA from load time to termination.

Since the TOS Language Processors require Monitor support, the Monitor Table is required. The Monitor Table is created the first time a PARAMETER or an EXECUTE command statement for a Class I program is encountered within a step and remains in physical core until step termination. When a user program is loaded, the Monitor Table is placed in Class 6 memory, preceding the loaded program. Changes to the Monitor Table caused by macros (ERFLC, STUT1, STUT2) are made to the Monitor Table in both Class 5 and Class 6 memory. Changes to the Monitor Table caused by the PARAMETER command are made to the Class 5 Monitor Table and to the Class 6 Monitor Table if a program is already loaded. The JCB has a pointer to the Monitor Table. This pointer is initialized to zero until the Monitor Table is created. When the Monitor Table is to be put into physical core, core is obtained, the address of the Monitor Table is placed in the JCB pointer, and the standard options are initialized. The Monitor Table is created only once during the step. At step termination, the Monitor Table is released.

The schematic for TCA control blocks and their pointers is shown in figure D-1. Although the TCA currently requires only 264 bytes, a total of 288 bytes is allocated to permit possible growth. This additional area is reserved and may not be used by the TOS Processor.

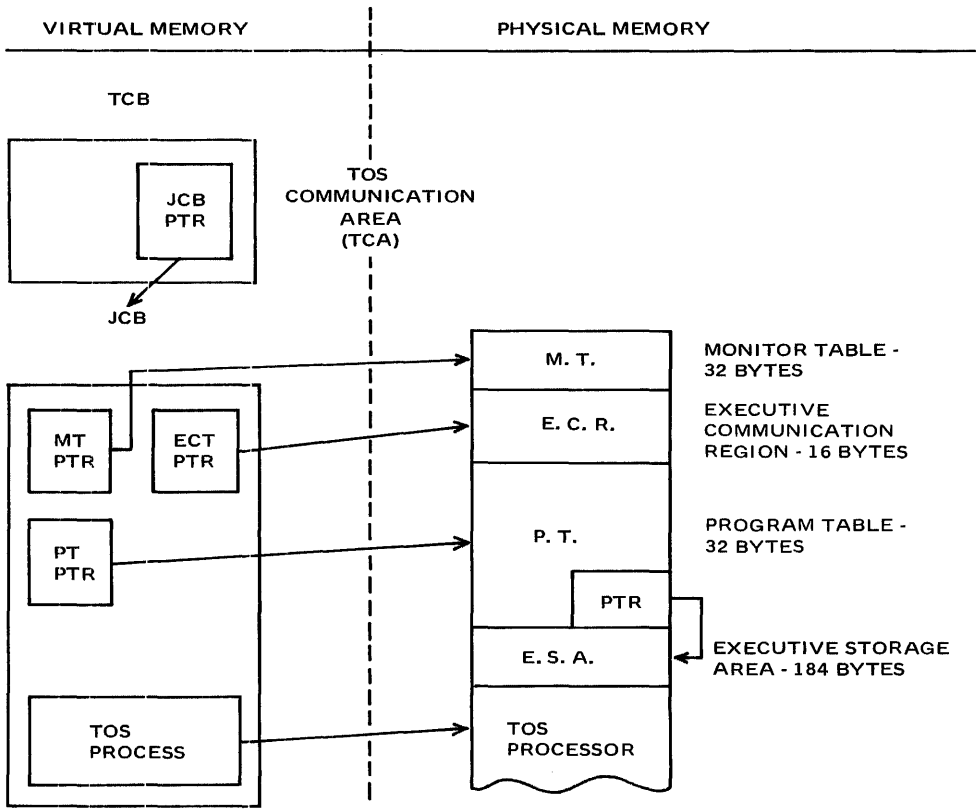


FIGURE D-1. TOS COMMUNICATION AREA (TCA) BLOCKS

MONITOR TABLE

Information retained in Monitor memory and required by programs running under Monitor control is found in the Monitor Table Area. The standard setting of the "yes" or "no" options at step initialization time are underlined in the table format.

The address of this area can be retrieved by using MONTB macro.

The information in this table is altered by the PARAMETER statement and specific macro calls. See table format that follows.

Byte	Bit	Information	Settings
0	2^7	PARAM TAPE option	0 = NO 1 = YES
	2^6	PARAM CARD option	0 = NO 1 = YES
	2^5	PARAM LIST option	0 = NO 1 = YES
	2^4	PARAM MAP option	1 = YES 0 = NO
	2^3	PARAM OBJLST option	0 = NO 1 = YES
	2^2	PARAM DIAG option	1 = YES 0 = NO
	2^1	PARAM XREF option	0 = NO 1 = YES
	2^0	PARAM DUPL option	0 = NO 1 = YES
1	2^7	PARAM DEBUG option	0 = NO 1 = YES
	2^{6-5}	PARAM CODE option	00 = EBCDIC 01 = 7094 10 = 3301
	2^4	PARAM WORK option (*)	1 = YES 0 = NO
	2^3	PARAM LIBRY option (*)	1 = YES 0 = NO
	2^2	PARAM ERRLST option	1 = YES 0 = NO
	2^1	PARAM ASMLST option	1 = YES 0 = NO
	2^0	PARAM INPUT option (*)	0 = SYSIPT 1 = Alternate tape
2	2^7	PARAM OUTPUT option (*)	0 = SYSUT1 1 = Alternate tape
	2^6	PARAM SOURCE option (*)	0 = SYSUT5 1 = Alternate tape
	2^5	SYSUT1 indicator	0 = No information present 1 = Information present
	2^4	SYSUT2 indicator	0 = No information present 1 = Information present

*Indicates options that may be eliminated or modified as the system is developed.

Byte	Bit	Information	Settings
	2^3	Subprocessor error flag	0 = No errors 1 = Errors
	2^2	Linkage Editor Call Indicator	0 = User Call 1 = Monitor Call
	2^{1-0}	Reserved	
3-8		Alternate source input device (*)	Symbolic
9-14		Alternate output device (*)	Symbolic
15-20		Bound program load name (*)	Symbolic
21-26		Alternate source output device (*)	Symbolic
27	2^7	PARAM DISC option	1 = YES 0 = NO
	2^6	PARAM ERRFIL option	0 = NO 1 = YES
	2^5	PARAM SYMDIC option	0 = NO 1 = YES
	2^{4-2}	PARAM SAVLST option	
		SAVLST = NO	000
		SAVLST = ALL	111
		SAVLST = SOURCE	100
		(or)	
		SAVLST = OBJECT	010
		(or)	
		SAVLST = LOCMAP	001
	2^8	PARAM FCPRTN option	
		FCPRTN = YES	X'00'
		FCPRTN = NO	X'FF'
		FCPRTN = nn	X'nn'

Note: nn represents the hexadecimal value of the COBOL FCP package to be included in the generated COBOL object program.

*Indicates options that may be eliminated or modified as the system is developed.

EXECUTIVE COMMUNICATION REGION (ECR)

Byte Nrs.	Entry Contains
0-8	Today's date MMDDYYDDD
9:2^7-2^5	Memory Size: always indicates 256K (101)
9:2^4	Memory Protection Option: always 1
9:2^3	System Timer Option: always 1
9:2^2	Direct Control Option: always 1
9:2^1-2^0	Processor: always indicates 70/46
10-11	(Device List Address: not supported in VMOS)
12-114	Not supported in VMOS

PROGRAM TABLE (PT)

Byte Nrs.	Entry Contains
0	Program Number
1	Reserved Space
2-7	Program Name
8	Program Priority
9:2^7	Monitor Subprocessor Entry Flag: always 1
9:2^{6-4}	Not supported in VMOS: always 0
9:2^{3-0}	Protection Key
10-15	Not supported in VMOS
16-19	Program Base Address
20-23	Program End Address
24-27	Address of ESA
28-31	Not supported in VMOS

Byte Nrs.	Entry Contains
0-3	Interrupt Mask Register
4-7	Interrupt Status Register
8-11	Program Counter
12-75	GR0 P1 through GR15 P1
76-107	FPR0, FPTR2, FPR4, FPR6
108-111	Prog. Base Address and Program size - 1
112-115	Not supported by VMOS: contains zeros.
116-119	Interrupt Weight
120-123	Address of Program Check Routine *(STXIT)
124-127	P1 P-ctr Stored on Program Check
128-135	GR10 P1, GR11 P1 Stored on Program Check
136-139	Address of Interval Timer Routine *(STXIT)
140-143	P1 P-ctr stored on Interval Timer Interrupt
144-151	GR10 P1, GR11 P1 stored on Interval Timer Interrupt
152-155	Address of Operator Communication Routine *(STXIT)
156-159	P1 P-ctr Stored on Operator Interrupt
160-167	GR10 P1, GR11 P1 stored on Operator Interrupt
168-171	Address of Unrecoverable Error Routine *(STXIT)
172-175	P1 P-ctr Stored on Unrecoverable Error
176-183	GR10 P1, GR11 P1 stored on Unrecoverable Error
172-183	(Status at CHKPT — not supported by VMOS)

*For VMOS, these contingency routine addresses must be specified by using the STXIT macro. VMOS Control System will update these addresses when a valid STXIT macro (SVC) is issued. When the VMOS Control System transfers to a user's contingency routine, the P1 P-ctr and GR10 P1 and GR11 P1 locations in ESA will be set by the VMOS Control System in the same manner as in TOS.

APPENDIX E
MACRO SVC IDENTIFICATION

This appendix lists the supervisory calls (SVC) for all user-available macros applicable to VMOS processing for which such calls are generated. These macros constitute the following three categories: VMOS Nonprivileged Macros, TOS Compatible Macros, and CAM Macros.

Macros are listed in this appendix in ascending hexadecimal order by SVC. The corresponding decimal equivalent for each SVC is also shown, as well as the category identifying the generic classification of each macro.

$SVC_{(10)}$	$SVC_{(16)}$	Macro	
1	1	TYPE	TOS Compatible
2	2	LPOV	
3	3	ADEXT	
4	4	STXIT	
5	5	DMODE	
6	6	DTYPE	
7	7	COMTY	
8	8	DDEV	
9	9	TERM	
11	B	EXCPW	
12	C	EXCP	
13	D	WAIT	
14	E	CKPT	
15	F		
16	10	ASCII	
17	11	EBCD	
18	12	EXIT	
19	13	EXIT	
20	14	EXIT	
21	15	SETIC	
22	16	TERMD	
23	17	GETOD	
24	18	GEPRT	
25	19	FLOAD	
26	1A	SMODE	
27	1B	SPRG	
28	1C	QUIET	
29	1D	ASSGN	
30	1E		
31	1F		
32	20	TOCOM	
33	21	EXCOM	
56	38	STUT2	
57	39	STUT1	
58	3A	PROUT	
59	3B	MONTB	
60	3C	ERFLG	
61	3D	WRTOT	
62	3E	RDCRD	
63	3F	TERMJ	

(Continued)

$SVC_{(10)}$	$SVC_{(16)}$	Macro	
66	42	RDATA	VMOS Nonprivileged
67	43	WROUT	
68	44	WRTRD	
69	45	WRLST	
70	46	TMODE	
71	47	SETSW	
72	48	GETSW	
73	49	REQM	
74	4A	RELM	
75	4B	CSTAT	
76	4C	PASS	
77	4D	CKPT	
78	4E	SPEXT	
80	50	GDATE	
81	51	ENDT	
82	52	EXITP	
85	55	EXECM,LOADM	
86	56	SETFB	
87	57	GTMAP	
89	59	VPASS	
90	5A	PDUMP	
91	5B	STOP	
92	5C	BKPT	
93	5D	MSG	
94	5E	ACCNT	
95	5F		
96	60	MSGNU	
106	6A	UNLOD	
107	6B	PRNT	
108	6C	PNCH	
109	6D	ITABL	
110	6E	LINK	
111	6F	TABLE	
147	93	FEOV	
148	94	GET(SAM)	
149	95	PUT(SAM)	
150	96	TYPIO	
151	97	GET(ISAM)	
152	98	PUT(ISAM)	
153	99	PUTX	
154	9A	SETL(ISAM)	
155	9B	STORE	
156	9C	OPEN	
157	9D	CATAL	
158	9E	ERASE	
159	9F	FILE	
160	A0	FSTAT	
161	A1	REL	
162	A2	CLOSE	
163	A3	GOTO	
164	A4	GETFL	
165	A5	FRS	
166	A6	RECON	
167	A7	LOG	
169	A9	RELSE	
173	AD	INSRT	
174	AE	ELIM	
175	AF	GETKY	
176	B0	GETR (ISAM)	
177	B1	EAM	

(Continued)

$SVC_{(10)}$	$SVC_{(16)}$	Macro	
178	B2	EXRTN,LBRET	VMOS Nonprivileged
179	B3	PAM	
180	B4	PTAM	
181	B5	CHNGE	
182	B6	COPY	
183	B7	READ,WRITE	
185	B9	GBUF	CAM
186	BA	FBUF	
187	BB	FETCH	
		CAMRD	
188	BC	WAITM	
189	BD	SEND	
190	BE	SHUTM	
191	BF	PRIME	
192	C0	FORM	
		NOLNS	
		LNTYP	
193	C1	SPSEQ	
194	C2	ACT,DEACT	

APPENDIX F
VMOS GENERAL SERVICE FILE MANAGEMENT MACROS

The four general service file management macro instructions available to the VMOS user are the CLOSE Macro, the FCB Macro, the IDFCB Macro, and the OPEN Macro.

CLOSE Macro

Indicates that file processing is complete and releases automatically-acquired buffer areas, performs label verification and creation, updates catalog entries (if required), ensures that pending write operations are completed, and positions volumes as indicated.

FCB Macro

Defines a file control block. FCB reserves space for a file control block and, optionally, supplies information to it. In addition, this macro reserves space to contain the logical routines needed to process the file.

IDFCB Macro

Provides symbolic name for an FCB. IDFCB is typically used to generate a dummy control section (DSECT) to provide symbolic names for the fields in an FCB.

OPEN Macro

Establishes a logical connection between a file and the problem program, completes the file control fields, verifies or creates file labels, positions volumes to the first record to be processed, and allocates buffer areas as required.

Additionally, the OPEN logic ensures that needed access routines are loaded and address relations are completed.

CLOSE Macro

The CLOSE macro instruction is used to disconnect a file from the user's program. During the execution of CLOSE, the user's trailer label routine will be given control if one was specified by the user. All I/O buffers automatically obtained by the system are also released. The FCB is reset to the contents which it had before the OPEN macro instruction was issued.

Name	Operation	Operand
[symbol]	CLOSE	{ALL / fcb-addrx / (1)} [,disposition-(0) {REPOS / RWD / DISCON / LEAVE}]

ALL

Specifies that all files for the task, except system related files (for example, SYSLST), are to be closed. User EAM files are also closed.

fcb

Specifies the address of the FCB associated with the file to be closed.

disposition

Specifies volume disposition for tape files (it is ignored for other devices).

REPOS

Positions the current volume to the beginning of the file.

RWD

Positions the current volume to BT.

DISCON

Rewinds the current volume to the disconnect point. Note that the device still belongs to the task.

LEAVE

Positions the current volume to the logical end of the file just processed.

Default: RWD

Programming Notes

1. For INOUT and OUTIN, if the last operation was not a WRT and WT or WRTWT, the LABEND exit is not taken. Furthermore, the system will not perform any label processing. This option is only applicable to BTAM.

2. If (0) is written, the hexadecimal disposition code is placed by the user in the low order byte of the register, as follows:

LEAVE	00
DISCON	01
REPOS	10
RWD	11

3. If (1) is written, the ALL option is indicated by inserting FF in the high order byte of the register.

4. Table F-1 details volume positioning for the REPOS and LEAVE options.

TABLE F-1. POSITIONING OF TAPE VOLUMES FOR OPTIONS OF THE CLOSE MACRO

Position 1		positions current volume to BT
Position 2	Labeled tape:	following tape mark that terminates trailer label group of file on current volume.
	Unlabeled tape:	following tape mark that terminates last data block of the portion of the file resident on the current volume.

Positioning of Tape Volumes:

OPEN TYPE	Position	
	LEAVE	REPOS
REVERSE	1	2
Other OPEN types	2	1

FCB Macro

The FCB macro – Define a Control Block (0) – instruction reserves space for a file control block and, optionally, supplies information to it. In addition, this macro optionally reserves space to contain the logical routines needed to process the file. Table F-2 indicates which FCB macros are allowed for each access method.

Name	Operation	Operands
SYMBOL	FCB	[LINK=linkname-symbol]
		[,FCBTYPE=ISAM|SAM PAM|BTAM]
		[FILE=filename-name]
		[,PASS=password-absexp]
		[,RETPD=retention period in days-absexp]
		[,RECFORM= {(F|V|U, A|M|N) / F|V|U}]
		[,RECSIZE=absexp]
		[,BLKSIZE=STD|(STD,absexp) |absexp]
		[,IOAREA1=NO|relexp]
		[,IOAREA2=NO|relexp]
		[,EXIT=(relexp)|relexp]
		[,IOREG=reg number-absexp]
		[,VARBLD=reg number-absexp] [,OVerlap=YES]
		[,KEYARG=relexp] [,KEYLEN=absexp] [,KEYPOS=absexp]
		[SHARUPD=YES|NO]
		[,LOGLEN=absexp] [,VALLEN=absexp] [,VALPROP=MAX|MIN]
		[,PAD=absexp] [,DUPEKY=YES]
		[,LABEL=STD|NSTD]
		[,OPEN=INPUT|OUTPUT|EXTEND| REVERSE|UPDATE|INOUT|OUTIN |SINOUT]
		[,LOGINFO=relexp]
		[,FORM=SHORT]

TABLE F-2. SUMMARY OF FCB MACRO INSTRUCTION PARAMETERS ALLOWED FOR EACH ACCESS METHOD

FCB Parameters	NULL Allowed	SAM	ISAM	PAM	BTAM	Parameters Allowed on FILE Command	Parameters that may be supplied from Catalog
LINK	X	X	X	X	X	X	
FCBTYPE	X	X	X	X	X	X	X
FILE		X	X	X	X	X	
PASS		X	X	X	X		
RETPD		X	X	X	X	X	
RECFORM	X	X	X	i	X	X	X
RECSIZE	X	X	X	i	X	X	X
BLKSIZE	X	X	X	i	X	X	X
IOAREA1		X	X	X	X		
EXIT		X	X	X	X		
IOREG		X	X				
VARBLD		X	i				
OVERLAP			X			X	
KEYARG			X	i			
KEYLEN	X		X	i		X	X
KEYPOS	X		X	i		X	X
SHARUPD	X		X	i		X	
LOGLEN	X		X	i		X	X
VALLEN	X		X	i		X	X
VALPROP	X		X	i		X	X
PAD			X	i		X	
DUPEKY			X	i		X	
LABEL		X			X	X	
OPEN		X	X	X	X	X	
FORM		X	X	i	i		

Legend: i — parameter accepted but ignored by action macros.
X — parameter may be specified.

Note: A null parameter is not the same as an omitted parameter. For example, omitting the FCBTYPE parameter will cause the macro to generate the default value for it, namely ISAM.

When the parameter is specified as null, (that is, FCBTYPE=,) the macro will be generated to request that the value of the null parameter be supplied by a subsequent FILE command/macro, or from the catalog entry for the file.

LINK=

Specifies a symbolic name of up to eight characters. This parameter provides the symbolic linkage by which data from a file command/macro may be used to modify an FCB. Thus, unique linkname(s) must be provided for the FCB(s) of a program when it is intended that these FCB(s) be modified.

When the linkname of an FCB is omitted, a linkname of spaces is generated. When LINK=, is specified, a linkname of binary zeros is generated. When either of these options is used, the FCB cannot be modified by a file command/macro.

FCBTYPE=

Specifies the access method to be used to process the file:

ISAM
SAM
PAM
BTAM.

Default: ISAM

FILE

Specifies the fully qualified filename (up to 54 bytes).

Default: FCB name (that is, symbol in the name field).

PASS=

Specifies the password required to access the file; the size of the password is ultimately 4 bytes. Note that this field is truncated or padded on the left in the manner which the assembler processes address constants; that is, the expansion is DC AL4 (password).

Example:

PASS=C'ABC'	yields a password of $(00C1C2C3)_{16}$
PASS=X'0102'	yields a password of $(00000102)_{16}$
PASS=C'ABCDEV'	yields a password of $(C3C4C5C6)_{16}$

Default: $(00000000)_{16}$

RETPD=

Specifies the period, in days, that a file is to be retained. This parameter stipulates when rewriting (or updating) of the file is permitted, but in no way is used by the system to automatically erase a file.

Default: 0 days, which allows for immediate rewriting or updating.

RECFORM=

F

Specifies fixed-length records.

V

Specifies variable-length records. This value is acceptable to BTAM but is treated exactly as a U specification.

U

Specifies undefined records. This value is not acceptable to ISAM.

Default: V

A

Specifies that the first data byte is an ASA print control character.

M

Specifies that the first data byte is a machine code (EBCDIC) control character. (Refer to Printer Reference Manual).

N

Specifies that no print control character is present.

Default: N

Although the PAM FCB accepts this parameter, it is ignored by the PAM action macros. Passing the parameter allows, for example, a file created by PAM to be processed by SAM. Note that SAM allows the user to specify a record format different from that specified when the file was initially created. For example, a file composed of format-F records could subsequently be read with format-U records specified. Consequently, a GET macro instruction would supply the set of logical records which can be contained in the buffer.

RECSIZE=

Specifies the length in bytes of a logical record for format-F and format-V records. If RECFORM=V, it is not necessary to supply this parameter: record size is assumed to be BLKSIZE. If the parameter is specified for format-V records, it specifies the maximum size record.

For format-U records, this parameter specifies the register (2 through 12) which will contain the length of each record. Whenever an undefined record is read, DMS supplies the length in this register. When an undefined record is to be written, the user must place the length of the record in this register.

Although the PAM FCB accepts this parameter, it is ignored by the PAM action macros. Passing the parameter allows, for example, a file, created by PAM to be processed by SAM.

BLKSIZE=

Specifies the size of the file's buffer.

STD

Specifies that the buffer is one half-page (2048 bytes).

(STD, absexp)

Specifies that a buffer is the number of half-pages specified in absexp. The maximum value of absexp is 32, which allows a BLKSIZE of 65,536. For ISAM files, the maximum value of absexp is 16, which allows a BLKSIZE of 32,768.

absexp

Specifies the size of a buffer in bytes. The maximum value is 4096.

Only BTAM and SAM (tape) may specify BLKSIZE=absexp; in fact, BTAM requires it. However, tape files processed by SAM can be specified with any form of the BLKSIZE operand. The primary intent of supporting these operand forms on tape is to provide real, yet reasonable, interchange of file processing between direct-access devices and tape.

Notes:

1. When this parameter is used at OPEN time (for example, to allocate buffers), BLKSIZE=STD is always assumed by the PAM action macros.

2. No logical record can exceed BLKSIZE. (No ISAM fixed-length logical record can exceed BLKSIZE-4.)

3. Strictly speaking, this parameter is buffer size (that is, the amount of transfer to or from an I/O device).

Example:

BLKSIZE=(STD,3).

The buffer is (2048) X (3)=6144 bytes; the physical block size is 2048 bytes. Note that a logical record can be up to 6144 bytes (which implies that logical records can cross physical blocks).

There are two principal reasons why the user might specify (STD,absexp) as contrasted to STD.

1. The logical record exceeds 2048 bytes.

2. The logical record size is inefficient for a block size of 2048. If the user had fixed length records of 1500 bytes and if BLKSIZE=STD is specified, then 548 bytes of each block is wasted. If the user specifies BLKSIZE=(STD,3) then 6000 bytes out of 6144 (that is, 2048 x 3) are used, which wastes 144 bytes per three blocks. Note that excessive concern with wastage may tend to increase the paging rate. For example, fixed-length records of 100 bytes waste 48 bytes per block, and only 44 bytes in a three-block buffer (6144-6100). This wastage is acceptable when the paging rate is considered. Generally, 10 to 15 percent wastage is acceptable.

When specifying the standard forms of the operand (BLKSIZE=STD or BLKSIZE=STD,absexp) the user does not consider the key. The key is always read from or written into the FCB.

Default: STD

IOAREA1=

No

Specifies that no buffer (area) is to be allocated by OPEN; this value cannot be specified for SAM and ISAM.

relexp

Specifies the address of the buffer (area) to process the file. If this area is less than or equal to a page (4096 bytes), it must be fully contained on one page and must begin on a word boundary. If this area is more than one page, it must:

1. Begin on a page boundary.

2. Be virtually contiguous.

3. Not cross a segment boundary.

Default: DMS dynamically acquires buffer space (class 5 memory) at OPEN time.

IOAREA2=

No

Specifies that no second buffer (area) is to be allocated by OPEN. The user is cautioned that if this value is specified for SAM, the system cannot buffer (overlap) I/O operations. The value cannot be specified for ISAM.

relexp

Specifies the address of the buffer (area) to process the file. If this area is less than or equal to a page (4096 bytes), it must be fully contained on one page and must begin on a word boundary. If this area is more than one page, it must:

1. Begin on a page boundary.

2. Be virtually contiguous.

3. Not cross a segment boundary.

Default: DMS dynamically acquires buffer space (class 5 memory) at OPEN time.

When a file is opened in a Class II program, the IOAREA addresses are created (if necessary) and verified. They are then moved to system memory. Accordingly, any change to these addresses in the FCB is completely ignored. New addresses can be effected only by closing the FCB and reissuing the OPEN.

This manner of processing is done to minimize overhead; that is, the system need not validate IOAREA addresses prior to each action macro call (on the assumption that they might have gotten changed). PAM and BTAM also allow the user to specify buffer areas in their action macros. These buffer addresses are, of course, validated each time. Additionally, PAM and BTAM allow the user to select IOAREA1 and/or IOAREA2 in their action macros.

During the period when a file is OPEN, the user must not disturb the contents of IOAREA in any manner other than that permitted in conjunction with the action macro processing. When a file is opened in a Class I program, IOAREA1 and 2 must be specified as relexp or NO.

If a SAM file is opened UPDATE, the IOAREA2 buffer is not used.

If NO is specified for IOAREA1, the value for IOAREA2 must also be NO. If IOAREA1 is specified as relexp, the value specified for IOAREA2 must be NO or relexp. If IOAREA1 is defaulted, then IOAREA2 must be also defaulted or be specified with a value of NO.

EXIT=

Specifies the address of an exit routine. If the value of this parameter is enclosed in parentheses, the address is indeed the address of the problem program's exit routine. If the value of this parameter is not enclosed in parentheses, the address is assumed to be the address of an EXLST macro instruction's code (refer to "Miscellaneous Tasks and File (Data) Management Commands and Macros" in the appendices).

It should be noted that if the exit is taken, regardless of its type, an indicator byte is set in the FCB so that the user routine can determine the specific exit condition. Moreover, if the exit is taken because of a hardware I/O abnormality, a 5-byte area in the FCB is also set to contain the following bytes from the CCB:

Standard device byte
Sense bytes 1, 2, 3
Executive flag byte

See table H1 in appendix H for descriptions of the Sense bytes.

The Executive flag byte and standard device byte are depicted in the expansion of the CCB DSECT, IDCCB.

Although the aforementioned points are discussed in depth in the EXLST macro instruction, two additional points are of note:

1. The OPEN contingency exit requires the EXLST macro instruction.

2. The COMMON parameter, supported by EXLST causes all nonspecified conditions to exit to the location specified in COMMON. If any other conditions (that is, other parameters) are specified, they are routed to their specified locations. In other words, COMMON allows programming by exception, thus freeing the programmer from the chore of writing a lengthy EXLST macro call and/or having the macro generate a long exit table.

Example:

EXLST OPENX=LOC1,COMMON=LOC

The OPENX condition causes control to be given to LOC1. All other conditions (for example, EOFADDR,DUPEKY) cause control to be given to LOC.

Default: No exit address is created; consequently, any abnormal conditions will cause abnormal program termination.

IOREG=

Specifies the register (2 through 12) which will contain the address of the current record. This implies records are to be processed in locate mode.

Default: Move mode is assumed.

VARBLD=

Specifies the register (2 through 12) which is to contain the amount of space remaining in the output area. This specification is required for SAM files when format-V records are created in locate mode.

Note: This parameter is ignored for ISAM.

OVERLAP=

Specifies that buffered or overlapped read operations are desired when more than one data buffer exists. This applies to the ISAM action macros GET and GETR.

Default: No anticipatory I/O is scheduled to provide overlap.

If this parameter is not specified, ISAM still uses IOAREA2. It is not used to perform anticipatory I/O, but rather as a work area.

KEYARG=

Specifies the address of the key. This is applicable to certain ISAM action macros (for example, GETKY, GETFL, ELIM).

KEYLEN=

Specifies the number of bytes in the record key. The key must be the same length in every record. The key may be from 1 to 255 bytes.

Default: 8

KEYPOS=

Specifies the position of the first character of the record key within the record. The key must be located in the same position of every record. The first byte of the record is 1. If format-V records are specified, the 4-byte length control prefix is considered to be a part of the record.

Default: 1 if record format is fixed; 5 if record format is variable.

SHARUPD=

Specifies whether an ISAM file may be shared by more than one task for the purpose of updating.

Default: NO

LOGLEN=

Specifies the length of the logical flag for a flagged ISAM file.

Default: 0

VALLEN=

Specifies the length of the value flag for a flagged ISAM file. KEYLEN+LOGLEN+VALLEN must be less than or equal to 255 if LOGLEN or VALLEN is specified.

Default: 0

VALPROP=

Specifies how the value of flags in a flagged ISAM file are to be propagated through the ISAM index structure.

Default: MIN

PAD=

Specifies the percentage of the buffer to be reserved during creation of the file for later insertion or expansion of records.

Default: Fifteen percent of the data space is reserved.

DUPEKY=

Specifies that duplicate keys are to be allowed.

Default: If duplicate keys are encountered, control is passed to the user's EXIT address.

LABEL=

Specifies label characteristics.

STD

Specifies standard labels.

NSTD

Specifies no labels or nonstandard labels (allowed for files on magnetic tape only).

Default: STD

All direct-access files must have standard labels. Moreover, user header and user trailer labels are not supported for direct access files.

OPEN=

Specifies how the file (FCB) is to be opened. Note that the OPEN macro instruction can override this specification. The permissible values are listed in table F-3.

TABLE F-3. PERMISSIBLE OPEN VALUES FOR ACCESS METHODS

OPEN Mode	SAM	ISAM	BTAM	PAM
INPUT	X	X	X	X
OUTPUT	X	X	X	
EXTEND	X	X		
REVERSE	X		X	
UPDATE	X			
INOUT		X	X	X
OUTIN		X	X	X
SINOUT			X	

FORM=

SHORT

Specifies that space for the appropriate logical routines is not to be reserved.

Default: Space is reserved where necessary. Note that PAM and BTAM never require logical routines in the user's program; accordingly, this parameter is ignored for these access methods.

CAUTION: The SHORT operand is for specialized use only. An FCB with the SHORT parameter cannot be used for actual ISAM and SAM processing. It is useful only as a prototype.

Programming Notes

When an FCB (file) is opened, most of the pertinent processing parameters are moved into system memory (P2 FCB) after their creation and verification. Any change to these parameters in the FCB may be ignored. New values should be effected only by closing the FCB and reissuing the OPEN. FCB changes to an opened file may cause abnormal termination or unpredictable file processing.

Parameters specified as null, for example,

FCBTYPE=,

are not given the default value. This provides a mechanism to note from the Assembly listing that a parameter is not completed at assembly time and must be specified at object time.

IDFCB Macro

The IDFCB macro - Provide Symbolic Name for an FCB (0) - instruction is typically used to generate a dummy control section (DSECT) which provides symbolic names for the fields in an FCB. Thus, with proper initialization of a base register, the user can access all fields of the FCB. (See figure F-1 for a method of accomplishing this base register initialization.)

Name	Operation	Operand
[symbol]	IDFCB	[D] [,prefix - {letter / *}]

D

Specifies that a DSECT card is to be generated.

Default: No DSECT card is generated.

prefix

Specifies the letter with which each symbolic name is to begin. If an asterisk is specified, then there is no prefix.

Default: Each name is prefixed with the letter I.

```
DFCBP 1     IDFCB
            .
            .
BEGIN       BALR 2,0
            USING * ,2
            USING DFCBP1,1      REG 1 COVERS IDFCB
            OPEN FILOT,FILOT,UPDATE
            .
            .
            .
            .
FILOT       FCB LINK=PAY              X
                  FILE=PAYFILE1       X
                  .
                  .
                  .
            END BEGIN
```

FIGURE F-1. BASE REGISTER INITIALIZATION FOR IDFCB

OPEN Macro

To activate an FCB(R)

Name	Operation	Operand
[symbol]	OPEN	{fcbname-addrx / (1)} [,mode-{code / (0)}]

fcbname

Specifies the address of the file control block. (See figure F-1 for the use of this parameter.) If (1) is written, the address of the FCB is placed in register 1.

mode

Specifies the type of OPEN desired. If (0) is written (indicating register 0), the mode code is placed in the low-order byte of the register, as follows:

Not Specified	$(00)_{16}$
INPUT	$(01)_{16}$
REVERSE	$(02)_{16}$
OUTPUT	$(04)_{16}$
EXTEND	$(08)_{16}$
UPDATE	$(10)_{16}$
INOUT	$(20)_{16}$
OUTIN	$(40)_{16}$
SINOUT	$(80)_{16}$

Default: The type of OPEN is specified in the FCB.

Note: The characteristics of the various types of OPEN are described under each applicable access method.

APPENDIX G
FILE MANAGEMENT DSECTS

DSECTS of file managements tables, parameter lists, FCB's and catalog entries are available to user programs through macro calls. The labels and alignment of the component fields are well defined; however, the arrangement of the regions is subject to change. The macro name and a brief description of each file management DSECT is given in table G1. Only those DSECTS of general interest to user (that is, nonprivileged) programs are listed.

It is suggested that the user, interested in current expansion of any particular DSECT, use the macro call with format as follows:

Name	Operation	Operands
[symbol]	Macro name	[D] [,prefix-{symbol / *}]

D

Specifies that the macro is to generate the DSECT pseudo instruction.

Default: The DSECT instruction is not generated.

Prefix

Symbol

Specifies the single character prefix to be concatenated to the beginning of all labels in the DSECT.

*

Specifies that no prefix is to be used.

Default: I

TABLE G-1. FILE MANAGEMENT DSECTS OF GENERAL INTEREST TO USER PROGRAMS

Macro Name	DSECT Description
IDBPL	BTAM Parameter List
IDCAT	CATALOG (CATAL) macro parameter list
IDCE	Catalog entry
IDCHA	CHANGE (CHAN) macro parameter list
IDCOP	COPY macro parameter list
IDEE	Catalog Entry extent list
IDEMS	File Management error messages
	This macro generates a list of EQU's describing the error messages output by file management modules. (See "File Management Error Message Concepts and Code Structure" in the appendices.)
IDERS	ERASE macro parameter list.
IDFCB	FCB (P1 region)
	Note that the DSECT describes all forms of the FCB.
IDFST	FSTATUS (FSTAT) macro parameter list
IDOST	OPEN/SHARUPD information area
IDPFL	FILE macro parameter list
IDPPL	Nonprivileged PAM parameter list
IDREL	RELEASE (REL) macro parameter list.
IDVT	VOLUME table entry

APPENDIX H
DEVICE MANAGEMENT CONTROL TABLES

PHYSICAL DEVICE TABLE (PDT)

The Physical Device Table (PDT) resides in Class I system memory. It consists of a series of physically contiguous elements, one element for each peripheral device connected to the System. The elements are ordered according to the VMOS device type code, in ascending sequence. The devices connected to the multiplexor channel are therefore at the beginning of the table. The System Drums(s) is the last entry in the table. The Console Typewriter and the remote terminal devices are not presented in this table.

The macro name for VMOS PDT is IDPDT.

CURRENT VOLUMES TABLE (CVT)

The Current Volume Table (CVT) also resides in Class I system memory. It consists of a series of physically contiguous 12-byte elements, one element for each volume which can be on-line to the system at any one time. If a device can support multiple volumes at the same time, the CVT elements for that device's volume will be contiguous in the CVT. The PDT element for that device will contain the address of the first CVT element in the table.

The prupose of the CVT is to reflect the Volume Serial Numbers of the volumes currently on-line to the System. In addition, the status of each volume is also maintained. If the volume is nonpublic, the task number of the owner-task is also carried in the CVT element.

The macro name for VMOS CVT is IDCVT.

COMMAND CONTROL BLOCK (CCB)

The Command Control Block (CCB) causes an area of storage to be reserved. This block is used by the programmer to provide the Executive routine with information required to effect an input/output operation. After the I/O operation has terminated, CCB is used by the Executive to report concerned status information. Refer to Table H-1 for the three sense bytes definitions for various peripheral devices.

The macro name for VMOS CCB is IDCCB.

TABLE H-1. DEFINES THE THREE SENSE BYTES FOR VARIOUS PERIPHERAL DEVICES

Sense Byte 1

Device Type	Bit 2^7	Bit 2^6	Bit 2^5	Bit 2^4	Bit 2^3	Bit 2^2	Bit 2^1	Bit 2^0
Magnetic Tape	Read or raw error	Service request not honored	*data block greater than count	Transmission parity error	Magnetic tape alarm	*beginning tape or end tape	*tape mark	Illegal operation
Card Reader	Read error	Service request not honored		Invalid punch code	*stacker select late			Illegal operation
Card Punch	Punch compare error	Punch Memory Parity Error		Transmission parity error				Illegal operation
Printer	*channel 12	*channel 9		Print line incomplete	Nonref code		End of forms	Illegal operation
Random Access	Read or raw error	Service request not honored	Seek check	Transmission parity error	Track check	Automatic head switching error	*end of file	Command code reject
Sense Byte 2								
Random Access	Count field data error	Overflow incomplete	Missing address markers	File protected	*not found	Invalid sequence	*end of cylinder	*track end
Sense Byte 3								
Random Access						Missing magazine (8568)		

Notes:

Those conditions not indicated by an asterisk result in setting of 2^7 and 2^5 of CCB byte 35.
Those conditions indicated by an asterisk result in setting of 2^7 and 2^6 of CCB byte 35.

APPENDIX J
COMMAND/MACRO NOTATION CONVENTIONS

GENERAL

A command constitutes an operation-oriented, system-supplied instructional entity. The number of required text lines to express the instruction is dependent on the number of operands constituting the command configuration and the use of continuation lines.

If cards are used, each card is treated as a line. A slash (/) must be in the first column of the first card of any set of cards comprising a command. Continuation cards must also contain the beginning slash.

If tape is used, the records are blocked in V-type format (variable length records). It is the user's responsibility to have this done. Maximum block size is 1024 bytes. Each record is a command and must begin with a slash (/) in its first position.

If commands are entered from a terminal, the user does not type a slash. The slash is typed by the Control Program when a command is expected. Continuation lines are entered by omitting the ETX at the end of the initial line and moving the write head and paper advance using the terminal's carriage return and line lead mechanisms.

Macro instructions are processed by the assembler using software-supplied macro definitions. VMOS macros are written according to the conventions described below; TOS macros do not normally follow these conventions because of compatibility requirements.

The processing of a macro instruction by the assembler is called the expansion of the macro instruction. The expansion results in fields of data and executable instructions, called the macro expansion. The fields of data, called parameters, specify the exact nature of the service to be performed and are contained in either registers (parameter registers) or data areas (parameter lists). If the parameters are contained in registers, only registers 0 and 1 may be used. If the parameters are contained in a parameter list, the address of that list is placed in register 1 and referred to by the called service routine.

MACRO INSTRUCTION LANGUAGE FORMAT

Command and macro instructions, like assembler instructions, are written in the following format:

Name	Operation	Operand

Name Field

The name field of the instruction may contain a symbol or be left blank. Normally, this symbol is the name associated with either the first executable instruction of the macro expansion, or in the case of commands, a locational identifier.

Operation Field

The operation field contains the mnemonic operation code of the instruction. This code may be a string of not more than eight alphanumeric characters, the first of which is alphabetic. For user written macros, the name should not begin with a $ since privileged system macros begin with this character.

Operand Field

The operand field may contain no operands, or one or more operands separated by commas. There are two types of operands: positional and keyword.

Positional Operands

When three or fewer operands are required by an instruction, positional operands are generally used. Positional operands must be written in a specific order. For instance:

EXAMPLE A, B, C

The assembly-time processing of operands A, B, and C is determined by the fact that they are the first, second, and third operands, respectively. If the second operand (B) is omitted, the user must supply the second comma so as to maintain the proper position for the third operand (C). Blanks may not be embedded in the positional field.

EXAMPLE A,,C

If the last positional operands are to be omitted, the delimiting commas need not be written. For example, if the operands B and C are to be omitted, the macro instruction could be written as follows:

EXAMPLE A

Keyword Operands

The keyword associated with a given keyword operand uniquely identifies that operand to the system. Therefore, these operands can be written in any order. A keyword operand is written as a keyword shown in each instruction description, immediately followed by an equal sign and its value. For instance:

EXAMPLE AREA =X,LENGTH=100

Mixed Operands

An operand field may contain both positional and keyword operands; however, all positional operands must precede all keyword operands. For Example:

EXAMPLE A,B,C, AREA=X,LENGTH=100

The rules for positional operand and keyword operand omissions apply to mixed operand fields. For example, if the operands B, C, and AREA from the above example are omitted:

EXAMPLE A,LENGTH=100

Operand Sublists (Macros Only)

A sublist consists of one or more positional operands, each separated by commas and the total list enclosed in parentheses. The entire sublist is considered to be one operand in the sense that it occupies a single position in the operand field or is associated with a single keyword. The contents of the sublist are processed similarly to positional operands.

The following operands are sublists:

(A,B,C)
(A)

Note, that in the second example, the sublist consists of only one operand. When a macro instruction description shows that an operand is to be written as a sublist, the enclosing parentheses must be written, even if there is only one element in the sublist.

Notational Symbols

Notational Symbols in the operand field of instruction descriptions assist the user in showing how, when, and where an operand should be written. The notational symbols are:

Vertical Stroke, shown as |; Braces { }; Brackets[ ]; Ellipsis, shown as . . . ; and Underscore.

1. Vertical stroke means "exclusive or". For example, A|B means that either the character A or the character B, but not both, may be written. Alternatives are also indicated by operands being aligned vertically, as shown in the next paragraph.

2. Braces denote grouping. They are used most often to group alternative operands. For instance, the following two operand descriptions are equivalent:

{INPUT | OUTPUT}

$$\left\{\begin{matrix}\text{INPUT} \\ \text{OUTPUT}\end{matrix}\right\}$$

3. Brackets denote options. Information enclosed in brackets may either be omitted or written in the macro instruction, depending on the service to be performed.

In the following case, the operand of the EXAMPLE macro instructions is optional and need not be supplied. However, if the operand is supplied, it must be one of the alternatives grouped in braces.

Name	Operation	Operand
[symbol]	EXAMP	[mode-{INPUT | OUTPUT}]

4. An underscore means that, if an operand is not specified, the underscored option is assumed (is the DEFAULT option). The underscored word INPUT in the above example indicates that, if the operand is omitted, INPUT is assumed.

5. The ellipsis denotes the optional occurrence of the preceding syntactical unit one or more times in succession. A syntactical unit is any combination of operand representations, commas and notational symbols enclosed in braces.

6. Upper-case (capital) letters indicate the portions which must be written exactly as shown. For example the operation field and coded values in the operand field must always be transcribed in upper-case letters.

7. Commas and parentheses must be written as shown in an operand field. They are delimiters, not notational symbols.

Value Mnemonics

Value mnemonics help the user remember the forms a particular operand may assume. There are 11 value mnemonics used in this publication. They are:

relexp
addr
addrx
addx
integer
absexp
value
text
code
symbol
characters

In the description of instructions in this publication, each positional operand is specified by a meaningful name hyphenated with a mnemonic as illustrated:

Name	Operation	Operand
[symbol]	EXAMP	name-value mnemonic

Each keyword operand is specified by the keyword, an equal sign, and a value mnemonic as illustrated:

Name	Operation	Operand
[symbol]	EXAMP	KEYWI=value mnemonic

Value Mnemonic Operand Forms (Macros Only)

For each value mnemonic, one or more operand forms may be substituted. For example, the value mnemonic, relexp, denotes that a relocatable expression may be written as the operand form; whereas the value mnemonic, addx, specifies that an explicit address or an implied address may be written.

There is a total of 10 operand forms; they are:

relocatable expression
register notation
explicit address
implied address
symbol
decimal integer
absolute expression
code
text
character

Table J-1 illustrates the value mnemonics and their permissible operand forms. In the following text, each operand form is fully described.

Relocatable Expression

A relocatable expression is one whose value would change by n if the program in which it appears is relocated n bytes away from its originally assigned area of storage. All relocatable expressions must have a positive value. A relocatable expression may be a relocatable term. A relocatable expression may contain relocatable terms – alone or in combination with absolute terms – under the following conditions'

1. There must be an odd number of relocatable terms.

2. All relocatable terms but one must be paired. Paring is described in "Absolute Expression."

3. The unpaired term must not be directly preceded by a minus sign.

4. A relocatable term must not enter into a multiply or divide operation.

A relocatable expression reduces to a single relocatable value. This value is the value of the odd relocatable term, adjusted by the values represented by the absolute terms and/or paired relocatable terms associated with it. The relocatability attribute is that of the odd relocatable term.

TABLE J-1. MACRO CONVENTIONS OPERAND FORMS

VALUE MNEMONICS

	Relocatable Expression	Register Notation	Explicit Address	Implied Address (Indexed)	Symbol	Decimal Integer	Absolute Expression	Code	Text	Characters
relexp	X									
absexp							X			
addr	X	X								
addrx		X	X	X						
addx			X	X						
integer						X				
value		X					X			
text									X	
code								X		
symbol					X					
characters										X

Note: An X indicates that the operand form may be written.

In the following examples of relocatable expressions, SAM, JOE and FRANK are in the same control section and are relocatable; PT is absolute.

SAM
SAM-JOE+FRANK
JOE-PT*5
SAM+3

Note that SAM-JOE is not relocatable, because the difference between two relocatable addresses is constant.

Register Notation

Register notation is written as an absolute expression enclosed in parentheses. The absolute expression, when evaluated, must be some value 2 through 12, indicating the corresponding general purpose register.

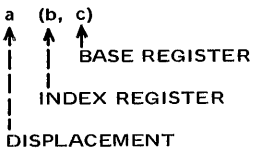
In the following examples of register notation, SAM and JOE are relocatable and PAL is absolute.

(5) indicates register 5
(SAM-JOE)
(PAL)
(PAL+3)

Explicit Address

The explicit address is written in the same form as an assembler language operand.

That is:



The following are examples of explicit addresses:

2(0,5)
0(2,4)

Implied Address (Indexed)

An implied address is written as a symbol, optionally indexed by a specified index register.

The following are examples of implied addresses:

GUPOFF
ALPMAY (4)

Symbol

The operand is written as a string of up to eight alphanumeric characters, the first of which is alphabetic. Embedded commas and blanks are not permitted. Symbols beginning with the characters $ may not be used. The symbols beginning with those characters are reserved for system use. The following are examples of symbols:

LEE
MARGIE3
BILL8SAM
DEBDEB

Decimal Integer

The operand may be written as a whole decimal number of one to eight digits, (for example, 5, 31, 127, etc.).

Absolute Expression

An absolute expression may be an absolute term or any arithmetic combination of absolute terms. An absolute term may be an absolute symbol or any of the self-defining terms. All arithmetic operations are permitted between absolute terms.

An absolute expression may contain relocatable terms alone or in combination with absolute terms under the following conditions:

1. There must be an even number of relocatable terms in the expression.

2. The relocatable terms must be paired. Each pair of terms must have the same relocatability attribute, for example, they appear in the same control section of an assembly. Each pair must consist of terms with opposite signs. The paired terms do not have to be contiguous, e.g., RT+AT-RT, where RT is relocatable and AT is absolute.

3. A relocatable term must not enter into a multiply or divide operation.

The pairing of relocatable terms (with opposite signs and the same relocatability attribute) cancels the effect of relocation. The value represented by the paired terms remains constant, regardless of program relocation. For example, in the absolute expression A-Y+X, A is an absolute term, and X and Y are relocatable terms with the same relocatability attribute. If A equals 50, Y equals 25, and X equals 10, the value of the expression would be 35. If X and Y are relocated by a factor of 100, their values would then be 125 and 110. However, the expression would still evaluate as 35 (50-125+110=135).

An absolute expression reduces to a single absolute value.

In the following examples of absolute expressions, JOE and SAM are relocatable and defined in the same control section, BERNY and DAVE are absolute:

331
DAVE
BERNY+DAVE-83
JOE-SAM
DAVE*4+BERNY

Code

A code is written exactly as indicated in the description of a macro instruction. For example:

Name	Operation	Operand
[symbol]	FTBAL	scores-code

scores specifies the desired action

TD - Touchdown
FG - Field goal
HT - Half time is called

The macro instruction could be written in a program:

```
SAM     FTBAL   TD
        FTBAL   FG
DUME    FTBAL   HT
```

Text

A text operand is written as a string of alphanumeric characters enclosed in apostrophes. Embedded blanks and special characters are permitted. Two apostrophes or two ampersands must be used to represent one apostrophe or one ampersand in the character string. The text operand may not exceed 255 characters including the enclosing apostrophes. For example:

'AREA,PCB,132, ,1256'

Characters

The character operand is written as a character string. Embedded commas or blanks are not permitted. Two apostrophes or two ampersands must be used to represent one apostrophe or one ampersand in the character string. The character string may not be enclosed in apostrophes. For example:

CUBTDAVE+HEINZ+JOHN*830PMOT

Oplist Operands (Macros Only)

In several macro instruction descriptions in this publication, the operand field is specified as:

Operand

oplist-{text / addr}

implying that a list of keyword and/or positional operands may be written as fields of a character string and that the character string itself (enclosed in apostrophes) or the address of the string may be written as the oplist operand depending on whether the text or addr form of the operand is chosen.

If oplist is presented as a character string, i.e., text operand form, the macro expansion places it in the assembled program followed by an end-of-message code, and loads a pointer to the string in register 1. If oplist is given as an address; i.e., addr operand form, the expansion places that address in register 1. In this case, the programmer must define the operands elsewhere in the program and provide an end-of-message code.

To reference and manipulate oplist macro instruction operands in coding, the address option of the macro instruction is used, permitting the set-up of the operand character string as a series of adjacent fields, each with its own label.

The string must end with a hexadecimal 27, which serves as an end-of-message code. Any unused space in each of the adjacent fields in the string must be filled with blanks to the maximum size of that field. Unlike other operand forms, all commas in an oplist operand must be written even if parameters are defaulted. A typical operand string might be coded:

```
OPLIST    DC    C'first operand'
OPLIST1   DC    C', second operand'
OPLIST2   DC    C',n operand'
          DC    X'27'
```

TYPE OF MACRO INSTRUCTIONS

R-Type Macro Instructions

Most system macro instructions are of two basic types: R-type (register) or S-type (storage). The letter R (R) or (S) follows the name of each macro instruction description in this publication to indicate its type. Macro instructions which are neither R- nor S-type, referred to as other macro instructions, are denoted by (0) in their descriptions.

Address operands in R-type macro instructions are always classified as addrx or addx. This arrangement allows the user to employ indexing although the addresses passed in R-type macro instructions must be properly covered; i.e., the base register used for the passed address must contain the proper value to ensure that the address refers to the desired location in virtual storage.

For example, assume there exists an R-type macro instruction RTYPE, which expects an address "area" in register 1 and the "length" of that area in register 0. Its external macro description would be as follows:

Name	Operation	Operand
[symbol]	RTYPE	{area-addrx / (1)} , {length-value / (0)}

Special Register Notation

The user's problem program might be written so that one or both of the parameters already exist in the proper parameter register when the macro instruction is issued. In this case, (1) or (0) is written as the operand. The notation (1) and (0) is referred to as special register notation. Register 1 and 0 cannot be used in a macro instruction unless special register notation is shown in the macro instruction description.

S-Type Macro Instructions

An S-type macro instruction is used when the number of parameters to be passed to the called routine cannot be contained in the two parameter registers. The parameters are placed in a parameter list whose address is passed to the called routine in register 1.

There are three forms of the S-type macro instruction:

1. The Standard Form

2. The L-form (Parameter list only).

3. The E-form (Executable code only).

> Note: All S-type macro instructions are provided with L- and E-forms unless otherwise stated under the individual descriptions.

The S-Type/Standard Form

The S-type/standard form macro instruction generates both the parameter list required by the called routine and the linkage to that routine.

Address operands in S-type/standard form macro instruction are always classified as addr. Hence, they may not be indexed, and the user's problem program is not responsible for providing cover registers.

As an example, assume an S-type macro instruction STYPE, which expects the address of two storage areas, "input" and "output" and the "length" of those areas. Its external macro description might be as follows:

Name	Operation	Operand
[symbol]	STYPE	input-addr,output-addr, length-value

The S-Type/L-Form

The L-form macro instruction is used to create a parameter list. E-form macro instructions are used to point to the parameter list that is generated by the L-form macro instruction.

The assembler recognizes an L-form macro instruction by the presence of the keyword operand MF=F in its operand field.

Because the L-form macro instruction generates only a parameter list, the use of operand types which require executable code, such as register notation, are prohibited.

There is an implied difference in the kinds of operands required in the external macro description, when using the various forms of the S-type macro instruction. Where the standard form indicates addr and value operands (register notation is allowed for example), it is implicitly understood that L-form macro instructions allow only relexp and absexp operands (register notation is not allowed for example).

The external description of the L-form STYPE macro instruction, previously described, becomes by implication:

Name	Operation	Operand
symbol	STYPE	input-relexp,output-relexp, length-absexp, MF=L

Note that the name field is required on the L-form because it usually becomes the label of the generated parameter list and is referred to by the E-form.

The L-form macro instruction generates the parameter list at the place the macro instruction is encountered. Because the L-form expansions contain no executable instructions, they should be placed in the program so as not to receive control; that is, among the DSs or DCs.

The S-Type/E-Form

A parameter list created by an L-form macro instruction, or by any other means, may be referred to by an E-form macro instruction.

The assembler recognizes an E-form macro instruction by the presence of the keyword operand:

MF=(E{list-addrx / (1)})

in its operand field. List should specify the location of the parameter list to be used by the E-form macro instruction. If (1) is written, register 1 should be loaded with the address of the L-form parameter list before execution of the macro instruction. The symbol in the name field of an L-form macro instruction becomes the name of the parameter list.

The external description of the E-form STYPE macro instruction, previously described, becomes by implication:

Name	Operation	Operand
symbol	STYPE	MF=(E, {list-addrx / (1)})

Other Macro Instructions

Certain system macro instructions cannot be classified as either R-type or S-type; they are referred to simply as other macro instructions, denoted (0) in the macro instruction descriptions.

Error Codes

Macros, which return error codes in a register, must place the code in the low order byte of register 15, the leftmost three bytes are cleared to zero.

Zero in the low order byte specifies that no error was encountered. Otherwise, the error code is specified as a multiple of four (to facilitate indexing).

Literals

Macros may use (generate) literals; in macros where the user is responsible for base-register coverage of operands, he too must generate literal coverage. Conversely, where the user does not have to provide base-register coverage, literals must be covered by the macros.

APPENDIX K
VOLUME TABLE OF CONTENT (VTOC) FORMATS

INTRODUCTION

This appendix describes the VTOC formats for all VMOS direct access volumes. Although VMOS VTOC formats are different from those of TOS/TDOS, some of the same terminology has been retained because the functions provided are similar:

Format 5 - defines which half pages are allocated and which are available. (See table K-2.)

Format 1 - (private volumes only) contains information describing the files which begin on that volume.

The VTOC consists of standard 2048-byte half-pages and always begins in the fourth half-page on the volume. The space that would be expected to contain the first three half-pages is not preformatted and contains the TOS/TDOS format IPL and SVL (Standard Volume Label).

The Format 1 labels contain an entry for each file that begins on the volume. Format 1 entries are exact duplicates of their corresponding catalog entries.

Table K-1 illustrates the VTOC format for all VMOS direct access volumes. Table K-2 illustrates the contents of the Format 5 half-page.

TABLE K-1. VTOC FORMAT FOR ALL VMOS DIRECT ACCESS VOLUMES

SYSRES volume	
page 1:	Bootstrap
page 2:	SVL
page 3:	Format 5 label
pages 4-6:	IPL
pages 7-Max:	Public space
Public volume	
page 1:	Dummy bootstrap
page 2:	SVL
page 3:	Format 5 label
page 4-Max:	Public space
Private volume	
page 1:	Dummy bootstrap
page 2:	SVL
page 3:	Format 5 label
pages 4-N:	Format 1 labels
page N+1-Max:	Private space

TABLE K-2. FORMAT 5 HALF-PAGE CONTENTS

Field	Size	Disp. from Beginning of Page	Description
1	2	0	Displacement of units table.
2	2	2	Displacement of packet table.
3	2	4	Displacement of segment table.
4	2	6	Total number of units available.
5	2	8	Total number of packets available.
6	2	10	Total number of files beginning on this volume.
7	2	12	Percentage of available packets.
8	2	14	Physical page number of the lhe page in the FMT 1 area (always = 5).
9	2	16	Physical page number of the rhe page in the FMT 1 area. (Significant on private volumes only.)
10	2	18	Number of bytes of significant data in F5.

	Device Type			Device Type			
	8564	8590	8568	8564	8590	8568	
11	128	508	1344	20	20	20	Units table
12	16	64	168	143	528	1364	Packet table
13	16	64	168	164	529	1532	Segment table
14	1868	1388	312	180	656	1700	Reserved

FORMAT -5 HALF-PAGE FIELDS

Field 1 – This 2-byte field contains the displacement of the first byte in the units table relative to the beginning of the page. Its value is the binary equivalent of $(20)_{10}$.

Field 2 – displacement of the first byte in the map table relative to the beginning of the page. Its value is the binary equivalent of:

$(148)_{10}$ for 8564

$(528)_{10}$ for 8590

$(1364)_{10}$ for 8568

Field 3 – displacement of the first byte in the contiguous packet table relative to the beginning of the page. Its value is the binary equivalent of:

164_{10} for 8564

592_{10} for 8590

1560_{10} for 8568

Field 4 – 2-byte binary value which defines the total number of available units on the volume. (1 unit = 3 half-pages)

Values are:

$8564 = 1013_{10}$

$8590 = 4058_{10}$

$8568 = 10920_{10}$

These values include room for 2 FMT 1 half-pages (5&6). If additional space for FMT 1's is requested by the user at volume initialization time, these values will be reduced accordingly.

Field 5 – 2-byte binary value defining the total number of completely empty packets on this volume.

Values are:

$8564 = 126_{10}$

$8590 = 507_{10}$

$8568 = 1364_{10}$

If additional FMT1 space is requested at Volume initialization time, these values may have to be reduced.

Field 6 – 2-byte binary count of the number of files beginning on this volume. Initially set = $00\ 00_{16}$.

Field 7 – 2-byte field containing the percentage of available packets on the volume. Initially set to $00\ 00_{16}$.

Field 8 – 2-byte physical page number defining the lhe of the FMT area. Its value is $00\ 04_{16}$. This value is present on both public and private volumes.

Field 9 – 2-byte physical page number defining the rhe of the FMT 1 area.

Field 10 – 2-byte field containing number of bytes in data portion of the F5.

Field 11 – Units Table - the units table defines the status of each unit on the volume (a unit is the smallest unit of allocation or 3 half-pages). A 1 bit indicates that the corresponding unit is available. A 0 bit indicates that the corresponding unit has been allocated. The length of the units table is dependent on the number of units on the volume. In all cases, however, the table is expanded to an integral number of words in order to facilitate searching.

8564 – There are 1015 possible units on the volume. The units table is 128 bytes long. Bits 0-1 are reset to 0 to indicate they are in use (1PL,SVL and VTOC). Bits 2-1013 are set to 1 to indicate they are not in use. Bits 1015-1023 are reset to 0 to indicate they are in use (they, in fact, do not exist).

8590 – There are 4060 possible units on the volume. The units table is 508 bytes long.

Bits 0-1 are set to 0 to indicate they are in use. Bits 2-4059 are set to 1 to indicate they are available. Bits 4060-4067 are reset to 0 because they represent nonexisting units.

8568 – There are 10,922 possible units on the volume. The units table is 1368 bytes long. Bits 0-1 are reset to 0 to indicate they are not in use. Bits 2-10921 are set to 1 to indicate they are available. Bits 10922-10943 are reset to 0 to indicate that they do not exist.

Notes:

Two important exceptions to the above definition are

1. If the user requests more than the minimum FMT 1 area at initialization, the table must be adjusted accordingly.

2. The space assigned to the alternate track pool must be marked as not available.

Field 12 – Packet table - defines the status of each packet (1 packet = 8 units) on the volume. There is one bit for each possible packet (each 8 units) on the volume. A 1 bit indicates that every unit in the corresponding packet is available. A 0 bit indicates that at least one unit in the corresponding packet has been allocated or is unavailable. The size of the packet table will be approximately 1/8 that of its corresponding units table (the packet table is rounded up to an integral number of full words).

Using the units tables defined above:

The map table for the 8564 would look like this:

Bit 0 = 0 (1st two units are allocated)
Bit 1-124 = 1
Bit 126 = 0 (last packet contains only seven units).
Bit 127 = 0 (these packets do not exist).

Field 13 – Segment Table - defines the largest number of contiguous empty packets in each segment (1 segment = 8 packets). There is one byte for each segment. The value of the byte will range between 0 and 8.

(0 means there are no completely empty packets in the segment – 8 means that every packet in the segment (8) is available.) This table is derived from the map table.

Example:

if the byte in packet table is:	the segment table byte would be:
$(01111110)_2$	$(06)_{16}$
$(01101110)_2$	$(03)_{16}$
$(10101010)_2$	$(01)_{16}$
$(11100111)_2$	$(03)_{16}$

The segment table for the 8564 would look as follows (assume the units and map tables defined above):

Byte 1 = $(7)_{16}$ – the first packet is not entirely available.

Bytes 2-15 = $(8)_{16}$

Byte 16 = $(6)_{16}$ – there are actually only 126 possible packets on an 8564.

APPENDIX L
FILE MANAGEMENT ERROR MESSAGE CONCEPTS AND CODE STRUCTURE

This appendix describes the conventions used by the file management components in the identification of errors by hexadecimal codes.

STRUCTURE OF CODES

All error codes generated by the file management routines are of the form

$(ORM)_{16}$

where R is the major component code, M the minor component code, and N the error number.

The major component code, (R) specifies the routine which last reported the error (that is, returned the code to the problem program). The major component codes are defined as follows:

2 – PAM (PQPAM)

3 – Catalog (DKCMSE)

4 – Allocator

5 – Commands/Macros

6 – Commands/ No Macros

8 – RESERVED

9 – Nonprivileged PAM (DQNPAM)

A – ISAM

B – SAM

C – BTAM (PDBTAM)

D – OPEN

E – CLOSE

The minor component code (M) specifies the routine which originally discovered the error. The minor component codes are defined as followed:

2 – PAM

3 – Catalog

4 – Allocator

5 – Commands/Macros

6 – Commands/No Macros

9-F – Used by all other components

The error number (N) is a unique identifier associated with the error.

Note that if M<9, then the original discoverer of the error was M. If M≥9, the original discoverer of the error was R.

CONSTRUCTION OF ERROR CODES

If a routine receives an error code from another routine, it replaces the major component code with its own; the rightmost two hexadecimal digits of the message are unchanged.

If a routine discovers an error, then it will construct a message consisting of its major component code, its minor component code, and a message number from 0-F.

NAMING OF MESSAGES

For ease of use, all messages are given a name which is equated to the error code. The list of all such names is maintained in the IDEMS macro which is described below. The names are of the following form:

DXY	,where
D	is the letter D.
X	is the prefix assigned to each DMS product.
Y	is a 1 to 5 alphanumeric character name.

Name	Operation	Operand
	IDEMS	[,ALL=Y] [,FMTER=Y] [,PAM=Y] [,CATAL=Y] [,ALLOC=Y] [,CMDMAC=Y] [,CMDNMAC=Y] [,OPEN=Y] [,CLOSE=Y] [,NPAM=Y] [,ISAM=Y] [,SAM=Y] [,BTAM=Y] [,P=prefix-code] [,PTAM=Y]

Where ALL=Y specifies all error codes to be generated. P is the prefix to be appended to each message. If not specified, I is used.

Each of the others specifies that the error codes for the indicated component should be generated.

RETURN OF CODES

The error codes will be returned to the calling program in either register 15 or the halfword D1ECB contained in the P1 FCB. This will depend on the linkage specified by the called routine.

Examples:

1. One level.

Suppose that a user calls nonprivileged PAM and the file is not open. Then nonprivileged PAM will place the error code 0994 in the FCB and exit via $GOTO to USERERR.

2. Two levels.

Suppose that a user calls nonprivileged PAM which in turn calls PAM which encounters an I/O error. Then PAM will return error code 0227 to nonprivileged PAM in register 15 and nonprivileged PAM will convert this to 0927, put it in the FCB and exit via $GOTO to ERRADDR.

3. Suppose that a user calls the allocator which calls $REQM and which calls PAM which calls $REQM. Then, the user may see the following error messages:

0444	$REQM returned error X'04' in register 15 to the allocator.
0424	$REQM returned error X'04' in register 15 to PAM which returned error 0224 to the allocator.

INDEX