WANG

VS

Operating System Services Manual

Operating System Services

3rd Edition — August, 1981

800-1107OS-03

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL: 617/459-5000, TWX 710-343-6769, TELEX 94-7421

Disclaimer of Warranties and Limitation of Liabilities

The staff of Wang Laboratories, Inc., has taken due care in preparing this manual; however, nothing contained herein modifies or alters in any way the standard terms and conditions of the Wang purchase, lease, or license agreement by which this software package was acquired, nor increases in any way Wang's liability to the customer. In no event shall Wang Laboratories, Inc., or its subsidiaries be liable for incidental or consequential damages in connection with or arising from the use of the software package, the accompanying manual, or any related materials.

NOTICE:

All Wang Program Products are licensed to customers in accordance with the terms and conditions of the Wang Laboratories, Inc. Standard Program Products License; no ownership of Wang Software is transferred and any use beyond the terms of the aforesaid License, without the written authorization of Wang Laboratories, Inc., is prohibited.

This Third Edition of the *VS Operating System Services Manual* (800-1107OS-03) documents Release 5.1 and succeeding releases until replaced or revised. It replaces and obsoletes the Second Edition (800-1107OS-02) and its Release 4 Addendum (800-1107OS-02.01). Changes are noted in the "Summary of Changes" and indicated in the text by single change bars in the margins.

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL: 617/459-5000, TWX 710-343-6769, TELEX 94-7421

Summary of Changes
for the 3rd Edition of the
VS Operating System Services
(800-11070S-03)

TYPE	DESCRIPTION	PAGES
NEW MACROS (AND SVCs)	FREEHEAP (SVC 57) (Deallocate Memory Block)	4-36/4-37, 6-79/6-80
	GETHEAP (SVC 56) (Allocate Memory Block)	4-39/4-41, 6-77/6-78
	LINKPARM Macro (Supply Program Parameters)	4-55/4-59
EXPANDED MACROS (and SVCs)	CHECK (TCIO & multiple event)	4-12/4-15
	CREATE PORT (Return 12)	4-17
	DISMOUNT (NODISPLAY option)	4-22
	DISMOUNT (SVC 41) (NODISPLAY option)	6-73
	EXTRACT (System, job, device, etc., data)	4-24, 4-26/4-31
	EXTRACT (SVC 28) (System, job, device, etc., data)	6-55/6-56
	MOUNT (NSA, NODISPLAY, NOMESSAGE)	4-61, 4-63
	MOUNT (SVC 30) (Return Codes)	6-58/6-59
	OPEN (WP PLOG option)	4-66
	READFDR (PLOG & PAREA options)	4-79
	READFDR (SVC 24) (WP files)	6-46
	RENAME (Renames libraries)	4-87
	RENAME (SVC 26) (Libraries)	6-48
	SET (JOB queue, class, CPU time)	4-97, 4-100/4-101
	UFBGEN (NODISPLAY, tape, PLOG)	4-123, 4-131/4-133
	XIO (UCPRINT, DEVSTATUS)	4-136, 4-139
	XIO (SVC 3) (Input parameters)	6-11
TECHNICAL REVISIONS	CXT (Macro example)	4-18
	Control Block Offsets	5-1/5-40
	XIO (SVC 3) (Tape and non-standard diskette)	6-13/6-14
	FREEBUF (SVC 6) (Interaction with GET/FREEHEAP SVCs)	6-19
	UNLINK (SVC 15) (Enhanced processing)	6-22
	CHECK (SVC 17) (IOSW in Regs 0 and 1)	6-25/6-28
	CHECK (SVC 17) (TC event)	6-26/6-28
	READVTOC (SVC 19) (Number of unused blocks)	6-29

TYPE	DESCRIPTION	PAGES
TECHNICAL REVISIONS	MOUNT (SVC 30) (Parameter extensions)	6-57
	MOUNT (SVC 30) (Non-standard addressing)	6-59
	PUTPARM (SVC 33) (Extensive revision)	6-23/6-67
	SET (SVC 35) (JOB parameters)	6-68
	DATANAME SUBBLOCK (FORTRAN fields)	B-18
EDITORIAL REVISIONS	CHECK (MESSAGE option)	4-14
	EXTRACT (Operand descriptions	4-26/4-28
	EXTRACT (SVC 28) (Parameter clarification)	6-52
	File Status Codes moved to Appendix C	7-48 C-12/C-19
	Miscellaneous Changes	4-25, 5-1, 6-21 6-53, B-15, B-17/B-20

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

CHAPTER 1: INTRODUCTION

The Wang VS Virtual Memory Operating System is intended as a low-overhead multiprogramming system for a medium-sized business machine configuration. As such, the design emphasis is on small size and simplicity of operation. This is apparent in many of the system's components: a simple scheduler, conventional I/O system, limited segmentation and memory protection capability, etc. On the other hand, facilities such as ANSI COBOL (1974), indexed file support, hierarchical file directory structures, and a large address space supported by an efficient paging mechanism make the system competitive with many larger computers.

The programmer should understand what the various system components are, how they interface with his program, what the (virtual) memory is used for and how it is partitioned for various purposes, and what certain operating system data structures (control blocks) contain. The purpose of this document is to provide this information.

Figures 1-1, 1-2, and 1-3 which follow are overall views of virtual memory and of the operating system's control block structure, and are provided for reference while reading the text of the document.

Segment 0 (SYSTEM SEGMENT - PROTECTED FROM USER MODIFICATION - ADDRESSABLE FROM ALL TASKS)

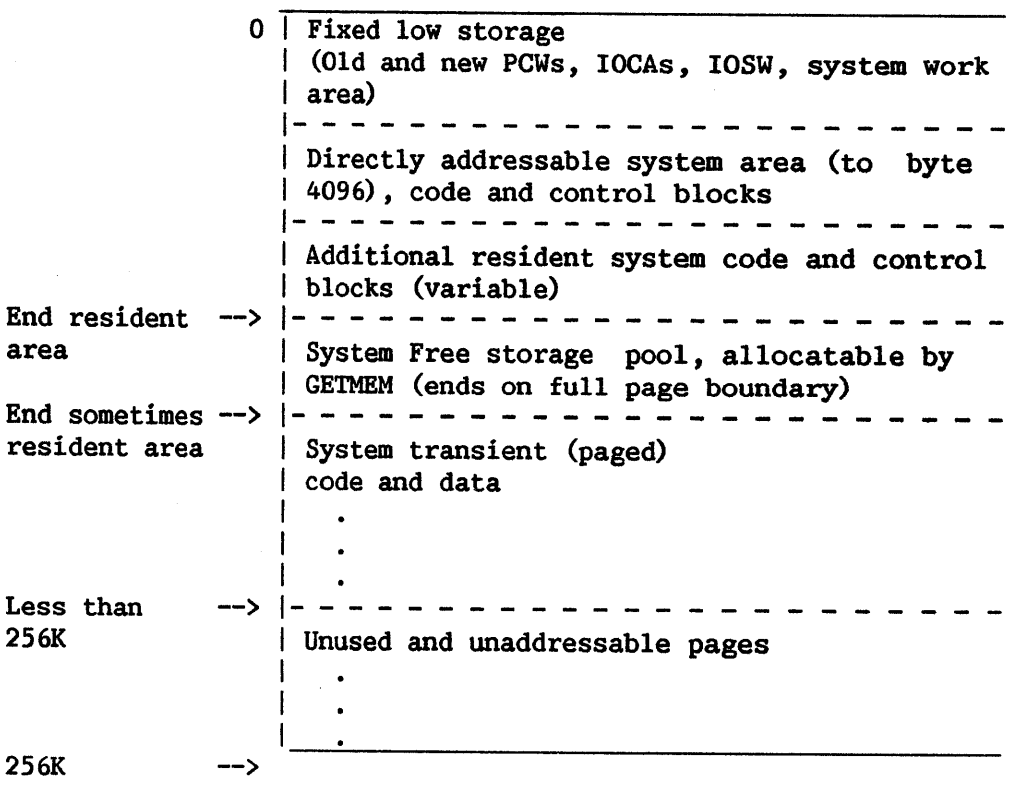


Figure 1-1. Virtual Memory Map - Segment 0 (System Segment)

Segment 1 (USER REENTRANT PROGRAM SEGMENT - PROTECTED FROM USER MODIFICATION - ADDRESSABLE FROM ONE OR MORE TASKS)

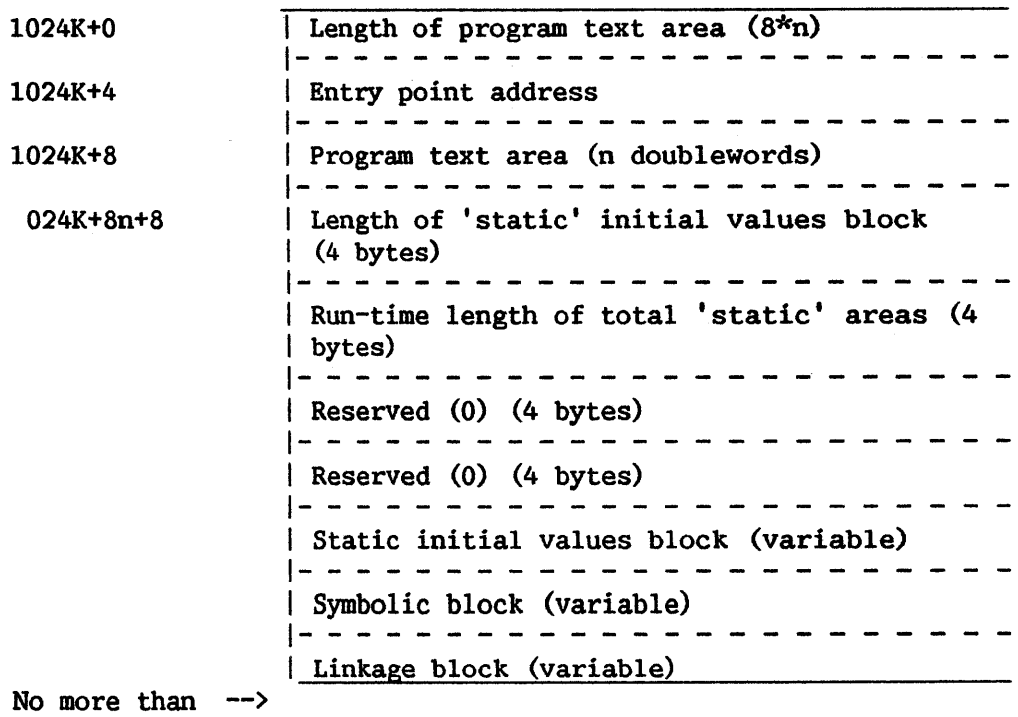


Figure 1-2. Virtual Memory Map-Segment 1 (User Program Segment)

Segment 2 (USER MODIFIABLE SEGMENT - NOT PROTECTED - ADDRESSABLE FROM ONE TASK)

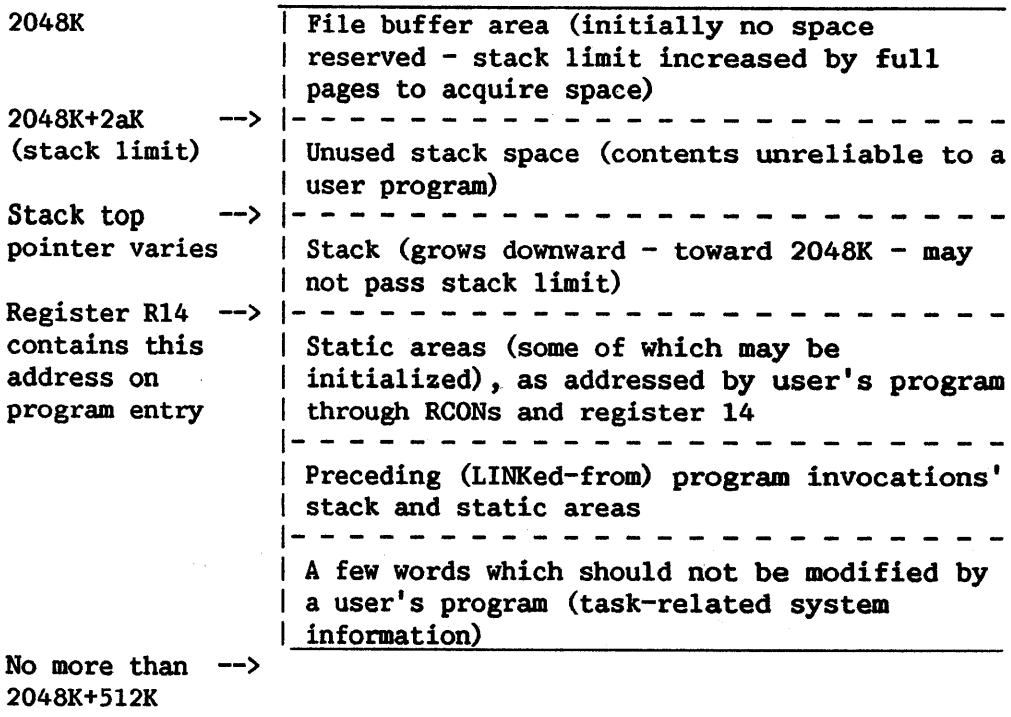


Figure 1-3. Virtual Memory Map-Segment 2(User Modifiable Segment)

CHAPTER 2: STANDARDS

2.1 PROGRAMMING

This section describes conventions which must be followed in writing code for the system in order to insure its successful execution and maintainability.

2.1.1 User Programs

These include system utilities which do not require special privileges, and data management routines which execute in an environment similar to that of normal user programs. Programmers writing entirely in a higher-level language (e.g., COBOL) need not learn all the details of these conventions, but should be aware of the standards for use of the workstation.

2.1.1.1 Constraints on Code and Data

For the following discussion, these definitions will be used:

Program - An entity either invoked by the command processor or by a LINK. This may be composed of several modules.

Object Program - The representation of a program on the disk after being processed by a compiler or linker.

Module - An individual assembled or compiled portion of a program. A program consists of one or more modules.

'Static' Area - This is the special area that has the property of appearing to be statically assigned space to a program using it. This space is allocated on a program basis and released on the same basis. The system will pre-initialize this area for the program with initial values specified in the object program.

2.1.1.1.1 The Reentrant Program Segment

User programs in segment one will be reentrant (may not modify themselves). All programs activated by the RUN command on the LINK SVC routine will be entered at the address specified in virtual location 1024K+4 (segment one, page zero, displacement four). Programs of up to 512K (K=1024) bytes (the full segment one) can be supported. The reentrancy requirement suggests the necessity of a separate modifiable area for variable data items, and the necessity of dynamically initializing variables in this modifiable area. This modifiable area is discussed in the following paragraphs. The only other constraints on a user program are that it must follow the standard conventions for making requests of the supervisor, transferring control between and within programs, and accepting and passing parameters. These conventions are described elsewhere in this document.

2.1.1.1.2 The Modifiable Segment

Although more than one user of the system may be sharing the code of the same program, each has a separate modifiable area. This area is organized as a single linear pushdown list (stack) which is entirely within segment 2 of the user program's virtual address space and which extends to address 2048K+n (n < 512K). The stack base (bottom-of-stack) is 2048K+n+1. The stack limit (lowest allowed stack location, addressed by control register 2) is 2048K plus a varying amount (between 0 and 128K) reserved for I/O buffers and other unprotected, 'heap'-allocated data areas. The stack grows downward. (See the VS Principles of Operation, Stack Oriented Instructions.) Proceeding downward, we have, on entry to a program:

(1) An unspecified number of bytes of system information, which should not be modified by the user program.

(2) An unspecified amount of information used by 'UNLINK' during final return from the program.

(3) 'Static' data areas as defined in the object format of the program to be initiated (see below).

(4) A save area containing register contents and a return point for final return from the program by means of the 'RETURN' macroinstruction or 'RTC 15'.

NOTE:

The 'stack base' is defined to be the next byte above (1).

Thus there may be many bytes of information on the stack when the user program is entered. Before the program uses any of the modifiable area other than the part containing this information, it must use one of the PUSH-type instructions to decrement general register 15 and thereby acquire additional space (this space being "pushed onto the stack"). Modifiable data items which are not to be treated in a stack-oriented manner should normally have space reserved for them at compilation time (by defining them in a 'static' area) or immediately on program initiation (either by pushing their initial values onto the stack or by reserving space with the PUSHN instruction), so that the rest of the stack may be used for stack-oriented data.

2.1.1.1.3 Creation of 'Static' Areas

The system supports a method for obtaining initialized or uninitialized 'static' memory on the stack. This support is invoked every time a new program is invoked by the RUN command or by a LINK, and not at any other time. Basically the support will assign an area of memory on the stack as the 'static' area. The size of the assigned area is determined by the size requested in the object module. Locations in this area that are to have initial values will then have them copied from the initial value section of the program's object module. Note that this allocation and initialization of memory will take place only when the program is started by the command processor or when the program is invoked by LINK. This means that program CALLs within the object program will not cause memory to be allocated or initialized.

When a program is initiated, either by the RUN command or LINK, register 14 will point to the lowest address of that object module's 'static' areas. Each program is expected to refer to particular 'static' areas through the following convention:

A special type of address constant (RCON) may be written in a program, naming a 'static' area, which is resolved to the displacement of that 'static' area from the address passed to the program in register 14. Thus the program can add the value in such an address constant to the contents of register 14 and then use the result to address a 'static' area.

When a LINK is performed, the issuer's 'static' areas are not changed. When the issuing program is returned,its initialized 'static' area contents are still available. The LINKed-to program, if it has a request in the object module for a 'static' area, will receive a fresh area on the stack. This area will remain for the LINKed-to program until it returns to the program which issued the LINK. Addresses of locations in the LINK issuer's 'static' areas will not be passed to programs invoked by LINK except as the program itself passes such an address. If LINK issuer wants, it can pass addresses in any of its 'static' areas, but a LINKed-to program cannot either pass back information in its 'static' areas or expect that values placed in its 'static' areas will be maintained between its invocations by LINK.

This system requires support on each of three levels:

(1) Compile or assemble time

(2) Linker (linkage editor) time (if binding is required)

(3) Program start-up time (either LINK or the RUN command initiating the program).

The support required at each step is:

Compile Time

The compiler or assembler has the responsibility to supply in the object program all information necessary to support this feature. To do this, the compiler will have to segment the program information and the data that is to be in a 'static' area. If the program is to be runable without having to bind (linkage edit), all references to addresses in 'static' areas will have to be specified as displacements into the block of 'static' areas for this object program.

When data is to be used from the 'static' area, the program will load a register with the value in the address constant which contains the displacement of the desired data, and will add the contents of register 14 to this value.

Binding Time

The linker (linkage editor) will collect the 'static' data sections from all the subprograms. If two or more subprograms have static sections with the same name, the initial values from the first of these sections will be used and the length of the resulting section will be the length of the longest of these sections.

Program Start-up Time

The same mechanism will be used with programs started by the command language and programs invoked by LINK. The mechanism will perform the services only at the time of the program invocation and not at any subsequent internal CALLs or program RETURNs.

At program start-up time, the system will:

a. Push onto the stack an area equal to the total size of 'static' sections as defined in the object program.

b. Copy values from the object program to the correct locations in each new 'static' section. This is not a simple move loop. The data is stored in the object program in a compressed format. It has values only for the areas that need initial values. Each initial value in the object program has with it a displacement into the whole block of 'static' sections.

c. Set register 14 to address the first byte of this area.

d. Perform additional program invocation processing.

2.1.1.1.4 Address Constants

There are three types of address constants permitted in a program: A, V and R type.

The A type is used for addresses of items within the same compilation or assembly. These may be the labels of instructions or constants, or may be addresses relative to either of these. Type A constants may be only in a program and refer to the program itself. Some examples are:

```
L     DC     A(L)     ADDRESS OF ITSELF
B     DC     A(L+2)   ADDRESS RELATIVE TO 'L'
```

The V type of address constant can be used to refer to locations known only by their external names. These names are normally in code areas or 'static' areas provided by another assembly or compilation.

Both these types (A and V) can be used in the code area (segment 1) to point to other code areas, but cannot be in the code area (segment 1) pointing to a 'static' area. (This restriction is caused by the constraint that no location in segment 1 can be set to a value differing from that on the program file from which it will be paged. The starting address of a 'static' area is unknown until the program starts running. For the address to be relocated in the program, it would involve modifying locations in the code area.) The system will, however, support address constants in the 'static' area that reference locations in either a code area or in other 'static' areas.

Example:

```
A    CODE              Name of code area (external name)
       .
       .
       .
B    DC      F'123'
C    DC      A(B)      Refer to location in same code area
     ENTRY   B         Makes B an external name
       .
       .
       .
D    CODE
       .
       .
       .
E    DC      A(B)      Refer to location in another code area
       .
       .
       .

F    STATIC            Name of 'static' area (external name)
     DC      A(B)      Refer to location in a code area
     DC      A(F)      Refer to location in this 'static' area
     DC      A(G2)     Refer to location in another 'static' area
       .
       .
       .
G    STATIC            Name of 'static' area (external name)
G1   DS      F
G2   DS      3F
     ENTRY   G2        Makes G2 an external name
       .
       .
       .
     END               End of this assembly

- - - - - - - - - - -

H    CODE
       .
       .
       .
I    STATIC
     DC      V(F)      Refer to a 'static' area in another assembly
     DC      V(G2)     Refer to a location in such a 'static'  area
     DC      V(B)      Refer to a location in a code area of
       .                 another assembly
       .
       .
     END               End of this assembly
```

The R type address constant is used by the program to locate 'static' areas. Because the program does not know the starting address of a 'static' area either when the program is compiled or linked, a method is needed to calculate the absolute address of a 'static' area (or of a location within a 'static' area) using a relative address in the program. When the program is initiated, register 14 will have the absolute address of the start of the block of static areas. Each R type address constant will contain the displacement into the whole block of 'static' areas (created by program initiation or LINK) of the named 'static' area or location within a 'static' area. The program will add the displacement in the R type address constant to the value in register 14. This will give the address of the item.

```
PROGR       CODE
              .
              .
              .
*                       INSTRUCTIONS TO ADDRESS A 'STATIC' AREA
            LR          R2,R14      STATIC BASE
            AL          R2,RDATA    PLUS STATIC AREA DISPLACEMENT
                                    GIVES ACTUAL ADDRESS OF
                                    'STATIC' AREA
            USING       DATA,R2
              .
              .
              .
RDATA       DC          R(DATA)     DISPLACEMENT OF 'STATIC'
                                    SECTION 'DATA'
DATA        STATIC
DATAITEM    DC          F'5'
            END
```

A more thorough example of assembler language source coding utilizing a 'static' area follows:

```
            REGS                    ,EQUATE REGISTERS
 PROG       CODE                    START A CODE SECTION
            BALR    R3,0            GET ADDRESSABILITY
            USING   *,R3
*
            L       R1,ADDWORD      GET ADDRESS OF CONSTANT
            L       R2,0(R1)        GET CONSTANT
*
*                                   FOLLOWING CODE USED TO
*                                   ADDRESS A 'STATIC' AREA
*
            LR      R2,R14          STATIC AREAS BASE
            AL      R2,RCON         ADDRESS AREA 'ST1'
            USING   ST1,R2
            ST      R3,DAT1         STORE CONSTANT IN 'STATIC' AREA
            SR      R0,R0           SET RETURN CODE TO ZERO
            RTC     15              RETURN TO CALLER
*
ST1         STATIC                  ESTABLISH 'STATIC' AREA NAMED
*                                   'ST1'
            DS      8F
DAT1        DS      F
*
PROG        CODE                    RESUME THE CODE SECTION
ADDWORD     DC      A(CONT)         ADDRESS OF DATA CONSTANT
CONT        DC      F'123'

RCON        DC      R(ST1)          DISPLACEMENT OF STATIC AREA
*                                   'ST1' WITHIN BLOCK OF ALLOCATED
*                                   STATIC AREAS
            END
```

2.1.1.1.5 Using The Assembler for 2200VS Object Code

While the 2200VS Assembler language is closely modeled after the IBM 360/370 Assembler language, there are some important differences, the most obvious of which is the addition of several instructions and the dropping of others. There are a few items that have changed in the conventions beyond the instructions.

To assemble a program, the following conventions are to be observed:

1. The first statement in a program should be a CODE statement. This will cause the text following it to be part of the reentrant program section named by the CODE statement. The CODE statement can be used the same as the IBM 360 assembler CSECT statement. The CSECT statement is not used on this system.

2. If a static area is desired, the STATIC statement should be used. The assembler allows any number of these statements and will allow initial values to be specified.

Rules for pseudo-instructions unique to the 2200VS are:

1. label CODE Not used; should be blank

   The label is required and must be a maximum of eight characters. It will be used as the external name of this reentrant program section. Note: ENTRY symbols in CODE or STATIC sections also are limited to eight characters.

2. label STATIC Not used; should be blank

   The label is required and must be a maximum of eight characters. It will be used as the external name of this 'static' section.

2.1.1.2 Transfer of Control

The modifiable area is also used in the course of transferring control between routines and programs in three ways:

(1) The BAL, BALR, BALCI, BALS or JSCI instructions may be used within a program to save a return point in a register or on the stack and enter a subroutine. The BC, BCR, BCS or RTC instructions are used to return.

(2) The SVC instruction is used to request services from the supervisor, and save the general registers on the stack before initiating the service routine. When the supervisory service has been performed, return to the user's program is effected by executing an SVCX instruction (in the supervisor). The user's program is concerned with nothing more than placing the address of required arguments (or occasionally the arguments themselves) on the top of the stack and issuing the SVC instruction. Routines entered by SVC instruction normally remove their input arguments and leave outputs on the top of the stack. Programmers interested in the details of the resulting stack manipulations may consult Figures 2-2 and 2-3. Figure 2-1 shows the associated register conventions which a user program must respect.

(3) The JSCI instruction is used to initiate a transfer of control between subprograms which have been bound together into a single program by the linker program. (Transfer of control between subprograms <u>not</u> bound together may be effected by LINK, as in (4) below.) The macroinstruction which generates the JSCI instruction is referred to as 'CALL'. Figure 2-4 shows the sequence of instructions (available as a macroinstruction) which is used to effect this transfer of control, and the resulting items pushed onto the stack.

(4) The LINK SVC is used to initiate a transfer of control between programs not bound together by the linker program. The LINKed-to program should return to the issuer by means of the same RETURN sequence (also in Figure 2-4) used in conjunction with CALL. The user should note that LINK will result in stack modifications which do not occur with CALL. See section 2.1.1.1.3, 'Creation of Static Areas', for a discussion of this.

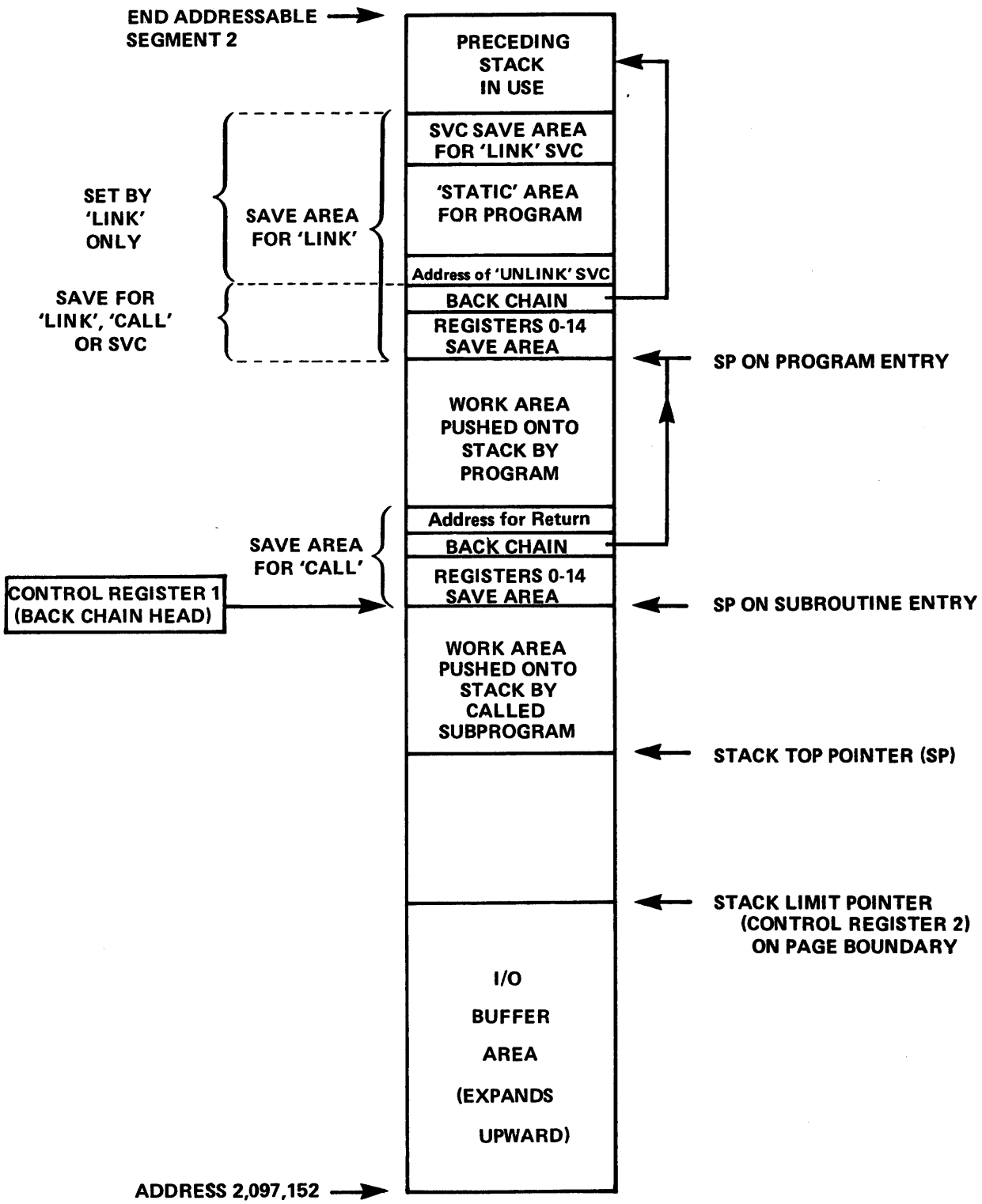
Registers

SP=15		Stack top pointer RESERVED (see Note 1)
R14=14		General (used to address 'static' areas)
R13=13	}	
.	}	
.	}	General use
.	}	(R1 is argument list pointer for use with CALL or
	}	LINK, by convention)
R0=0	}	

Note 1. Register 15 (SP) must always address the lowest location on the stack which contains usable data or into which data may be placed by any non-PUSH-type instructions. This constraint is imposed on all programs.

Figure 2-1.

Figure 2.2.


END ADDRESSABLE SEGMENT 2
PRECEDING STACK IN USE
SVC SAVE AREA FOR 'LINK' SVC
'STATIC' AREA FOR PROGRAM
Address of 'UNLINK' SVC
BACK CHAIN
REGISTERS 0-14 SAVE AREA
WORK AREA PUSHED ONTO STACK BY PROGRAM
Address for Return
BACK CHAIN
REGISTERS 0-14 SAVE AREA
WORK AREA PUSHED ONTO STACK BY CALLED SUBPROGRAM
I/O BUFFER AREA (EXPANDS UPWARD)
ADDRESS 2,097,152
SET BY 'LINK' ONLY
SAVE AREA FOR 'LINK'
SAVE FOR 'LINK', 'CALL' OR SVC
SAVE AREA FOR 'CALL'
CONTROL REGISTER 1 (BACK CHAIN HEAD)
SP ON PROGRAM ENTRY
SP ON SUBROUTINE ENTRY
STACK TOP POINTER (SP)
STACK LIMIT POINTER (CONTROL REGISTER 2) ON PAGE BOUNDARY

Stack linkage item after SVC entry:

Byte	
68	PCW (status)
64	PCW (return) with SVC number
60	Back chain to next outer-more SVC save area or CALL/LINK save area
56	Reg 14 save
52	Reg 13 save
48	Reg 12 save
0-47	Regs 0-11 save

Entry to SVC: SVC (SVC#)

Exit from SVC: SVCX (Register)

Figure 2-3.

Stack linkage item after CALL:

Byte

64	Address of return point
60	Back chain to previous CALL/LINK save area
0-59	Regs 0-14 save

```
CALL:     JSCI     15,=A(Entry-point)

RETURN:   LA       0,return-code
          RTC      15 (if unconditional)
```

Note that the value of register 0 saved by JSCI is not restored to the register by RTC.

Figure 2-4.

CALL

Linkage by: JSCI condition, indirect-word-address

Exit by: LA 0,Return-code
RTC Condition

Registers	BEFORE	ON ENTRY	BEFORE EXIT	AFTER RETURN
R0	Irrelevant	As BEFORE	Irrelevant	Return-code
R1	Argument list pointer	As BEFORE	Irrelevant	As BEFORE*
R14	'Static' base pointer**	As BEFORE	Irrelevant	As BEFORE*
SP	Stack top	Updated stack top	Irrelevant	As BEFORE*
Other	Irrelevant	As BEFORE	Irrelevant	As BEFORE*
(Control register 1)	Previous save area pointer	New save area pointer (same as SP)	As ON ENTRY	As BEFORE

* Unless current save area is modified before return.

** Not enforced by the system. Register 14 may be used for other purposes by the user program if appropriate.

Figure 2-5.

LINK

Linkage by: SVC LINK

Exit by: LA 0,Return-code
RTC Condition

Registers	BEFORE	ON ENTRY	BEFORE EXIT	AFTER RETURN
R0	Irrelevant	As BEFORE*	Irrelevant	Return-code
R1	Argument list pointer	As BEFORE	Irrelevant	As BEFORE
R14	'Static' base pointer*****	New 'static' base pointer	Irrelevant	As BEFORE
SP	Stack top (LINK parameters)	Updated stack top***	Irrelevant	As BEFORE****
Other	Irrelevant	As BEFORE	Irrelevant	As BEFORE
(Control register 1)	Previous save area pointer	New save area pointer (same as SP)	As ON ENTRY	As BEFORE

* Program entry point address in register R0 if LOADONLY option specified.

*** New 'static' areas, UNLINK information, register save area, return address on stack.

**** But LINK parameters popped from stack (see SVC LINK description).

***** Not enforced by the system.

Figure 2-5. (continued)

2.1.1.3 Supervisor Interface

As mentioned briefly above, the SVC instruction is used as the normal interface between user programs and supervisory routines. In addition to the effect of the supervisory service request saving status information on the stack, as in Figure 2.3 above, there are four points which the writer of SVC routines and of system programs in user program mode should be aware of:

(1) The top word of the stack should address an argument list when the SVC is executed if the arguments for the particular SVC are sufficiently many or sufficiently long that passing them directly on the top of the stack is infeasible.

(2) Register 15 (SP) is generally the only register which must contain useful information when the SVC instruction is executed.

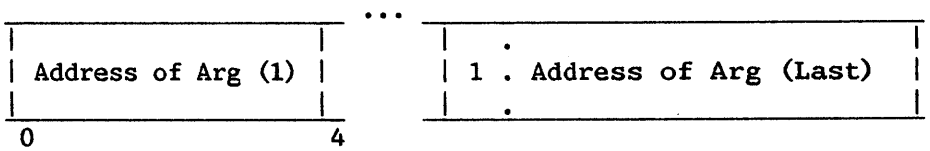
(3) Corresponding to most valid SVC numbers are macroinstructions which may be used in assembly language programming to generate the SVC and pass its arguments. These macroinstructions are described in Chapter 4 of this document.

(4) All registers except register 15 (SP) are restored on return from an SVC routine. When an argument list is provided, results may be returned in one of the arguments. In order to do this, the argument list must be in a user's modifiable data area or in the system area (segments 2 or 0). Otherwise results are returned on the top of the stack, replacing all input arguments.

2.1.1.4 Standard Argument Lists for CALL or LINK

Communication between modular sections of programming is normally effected through an argument list in standard format. Such a list is constructed to pass parameters to a subprogram invoked by the standard CALL sequence or LINK supervisor call.

The format is as follows:



Arg(1) through Arg(Last) are four-byte address values pointing to the locations of the actual arguments. The high-order bit of the last four-byte area should be set to one to indicate the end of the list. Exceptionally, the arguments themselves may be placed in the list in place of their addresses. Register R1 is loaded to address this list before control is passed to the receiving routine. Before initially invoking a program, the program initiator (command processor) will set register R1 to binary zeroes to indicate that no parameter list is being passed to the program.

Note that a user program may return result values in an argument list only if it can be guaranteed that the argument list is in the user's modifiable area.

2.1.1.5 Use of Program Level Parameters

System conventions for communication between the user's program interface (i.e., the command language) and the invoked programs are not compatible with the use of standard argument lists. To be runnable from the command language, programs must access all run-time parameters using the GETPARM SVC. The GETPARM facility is used to access run-time parameters in groups of keyword-identified values which are interpreted as:

(1) Run-time device or file assignments,

(2) Batch oriented run-time option lists, or

(3) Interaction oriented program command data.

Programs which require a more flexible level of interaction than that provided by GETPARM should access the user workstation using the standard data management facilities.

Parameters requested using GETPARM are usually accessed through direct interaction at the user workstation. Alternatively, these parameters can be prespecified within procedures using the parameter specification statement. In addition, the PUTPARM macroinstruction is available to allow a program to supply parameters which are then available through GETPARM to the next subprogram LINKed to by this program.

2.1.1.5.1 Run-Time Device and File Assignment

Parameter groups solicited by the OPEN routine are used to assign actual devices or files to the internal filenames used within programs for data transfer. All action related to device/file allocation, file lookup, and control block generation is performed by OPEN. OPEN uses a number of parameters which must be specified by the union of information supplied in the User File Block (the only explicit OPEN parameter) and the information obtained using GETPARM. The User File Block is compiled into the user's program.

Some of these parameters are suppliable only in the User File Block (UFB). The other parameters are solicited using the GETPARM facility with all solicited parameters defaultable to values which can be supplied in the UFB. GETPARM enables the user (or the command language) to modify the default values specified; thus the GETPARM supplied values override UFB information.

2.1.1.5.2 Default File Specifications

Default information for the specification of actual files at run-time can be placed in the UFB anytime before a file is OPENed. This facility should be used to minimize the amount of information which must be acquired from the user. Information which can be defaulted includes File Location Information and File Size.

The disk space requirements of all output files (including 'print' files) should be specified in the UFB if possible. In general, the size of an output file is related in some manner to the size of an input file. Input file size is available in the UFB for any OPEN input file located on a disk device. Thus, when input files can be OPENed first, this size information may be used to calculate default space requirements to be placed in the output file UFB.

All information necessary to specify work files should be presupplied in the UFB or elsewhere. Space requirements may be developed in the manner described for output files. File location information, consisting of a filename and a volume designation, should be developed as follows. The volume location of the work files can be left blank in the UFB, to be supplied in the ETCB by means of a command language 'SET' command. The library name for work files is ignored in the UFB before OPEN. It is set to a special user work library (associated with the logged-on user) by OPEN processing. The default actual filename should be formed by using characters '#' or '##' as a prefix to a maximum four-character name which is unique to the program. OPEN will place after the supplied name a four-character suffix sufficient to guarantee temporary uniqueness. Work files whose names begin with '#' are scratched when closed. Those whose names begin with '##' are retained until run termination.

2.1.1.5.3 Run-Time Specification of Options

Batch oriented lists of run-time options should be solicited directly by the programs using the GETPARM facility. By convention, the parameter-reference-name OPTIONS should be used to identify this type of parameter group. Also by convention, users of this facility are asked to specify reasonable defaults for all keywords specified.

2.1.1.5.4 Run-Time Command Interaction

Programs which are inherently interactive may be able to use GETPARM to solicit user directives and associated parameters. Each format presented to the user must be labeled with a parameter-reference-name. The programs using this facility can be invoked and directed from procedures.

2.1.1.5.5 Standard Parameter-Reference-Names

The parameter-reference-names (PRNAMEs) used to identify file specifications and other parameter groups solicited through GETPARM should be chosen to assist the user in easy identification of parameter function. Within groups of related programs, naming conventions should be established to enhance recognition and predictability.

Standard PRNAMEs will be assigned for system utility programs and compilers. The PRNAMEs which have been assigned to date are the following:

INPUT	- The basic input file (if only one).
INPUT1-INPUT(n)	- Multiple input files used for equivalent purposes.
OUTPUT	- The basic output file.
LIBRARY	- A library used for input purposes.
WORK	- An I/O file used for temporary storage (if only one).
WORK1-WORK(n)	- I/O files used for temporary storage.
OPTIONS	- The batch oriented option list used to define run-time parameters.
DISPLAY	- The user's workstation.
PRINT	- An output file containing data to be printed.

2.1.2 Supervisor Call Routines

Routines are entered by SVC instructions to accomplish functions which require:

- Modification of the system area of memory, which is normally protected from the user, or
- Execution of privileged instructions, or
- Intimate knowledge of the operating system's organization,

and which are not accomplished by non-supervisor-call coding in dedicated system tasks such as the paging task. Among these functions are:

- Initiation of physical I/O requests.
- Opening and closing files, including allocation and construction of Open File Blocks.
- Synchronizing tasks through semaphores.
- Performing service functions for use by other supervisor call routines (such as allocating system area memory space for control blocks).

The system programmer should note that the special 'paging task' cannot in the current implementation issue SVC instructions.

Supervisor call routines accept parameters on the top of the stack. The lowest parameter address is at displacement 72 (decimal) from the stack top pointer on entry. They are entered by means of the SVC instruction, which establishes a new save area on the task's system stack as shown in Figure 2.3. These routines may extend the stack in the user's modifiable area by PUSHing items onto it. They need not POP all these items off again before returning. Supervisor call routines should exit by issuing an SVCX instruction, which restores all registers and the Program Control Word from the user's stack and sets the stack top pointer (register 15) to the address value in the register specified in the R1 field of the SVCX instruction.

2.1.3 System Dialogue and Workstation Use

The workstation of the Wang 2200VS System must share the duties of:

(1) the primary data-entry and display device used in a program

(2) a system console device for presentation of and response to system messages, which may be:

   (a) solicited (e.g., responses to commands)

   (b) unsolicited (e.g., volume mounting requests, device error information, etc.)

Special conventions and protocol are necessary to meet the requirements of these multiple uses:

(1) Supervisor call routines and unprivileged system code (such as data management routines) must use the CANCEL or GETPARM supervisor calls to send messages to the workstation.

(2) Messages must be in the format specified in the descriptions of CANCEL and GETPARM in Chapter 6 of this document.

(3) Messages relating to background program runs (programs running without an associated workstation) will be sent to the system console device.

(4) When GETPARM processing is entered, the workstation screen, its resident buffer contents, its status (keyboard locked or unlocked, 'attentions' received, etc.) and its tab settings are saved. Before resuming user program processing, this screen and status are restored. This saving and restoring also occurs when the 'Help Processor' is entered as a result of the user striking the HELP key on his workstation.

2.1.4 Error Handling

There are several classes of errors which the operating system may encounter, and which require differing responses:

- Program exceptions in user programs
- Program exceptions and other uncorrectable errors in system routines
- Invalid parameters passed to system routines
- I/O errors

(1) Program exceptions in users' programs are of concern to the operating system's program check interrupt handler, which passes control to the HELP processor or to a program-specified exception handler.

(2) Program exceptions and other uncorrectable errors in system routines may result in message display by the CANCEL SVC (disallowing continuation of the program's processing) if their cause can be traced to the user's program (as may be the case with certain exceptions in supervisor call routines). Otherwise they must be considered probable system failures, and result in an orderly system halt.

(3) If a system routine receives invalid parameters, it should be able to detect this during initial validation of the parameters and before using them for further processing. It then responds as in (2) above.

(4) I/O errors may be of three kinds:

- Soft errors, signifying that an I/O operation was successfully completed after retry by the separate I/O Processor (IOP).
- Hard errors, signifying failure of an I/O operation (including memory parity errors detected on an I/O operation).
- Logical file processing errors which do not reflect any errors occurring during an actual I/O operation.

Soft error indications are passed to the CHECK supervisor call routine (in the I/O Status Word) to be logged in a system error logging file, and otherwise ignored. Hard errors are passed in the same way. The task responsible for the I/O either issues a CHECK to wait for completion of the associated I/O request or will make reference to the file again (by another Data Management Function request, a CLOSE, or an implied CLOSE on program termination). At that time, error indications are examined and the user's I/O error routine entered for hard errors if such a routine has been provided. In the absence of such a routine, the user's program may be abnormally terminated at the discretion of the Data Management routines, through issuance of a CANCEL supervisor call. Memory parity errors detected by I/O processors are logged just like other I/O errors.

The user's I/O error processing routine and his end-of-data and invalid-key-condition routine are specified in the User File Block (see Chapter 5). These routines are entered after interpretation of the I/O Status Word (IOSW) by the Data Management System routines. The I/O error routine is entered on logical file processing errors (such as invalid function requests) as well as on actual hard I/O errors.

These routines are entered from the unprivileged data management routines as if the data management function had returned normally, but with the return address modified to be one of these addresses from the UFB. All register contents are restored except register zero, which is set to contain the normal return address from the function (next instruction after the JSCI instruction by which the function was initiated). Register 1 continues to address the User File Block, which may be used to determine the nature of the unusual occurrence, as indicated in fields UFBFS1 and UFBFS2 of this block (file status bytes). The I/O error processing routine is entered for all unusual conditions, including end-of-data and invalid-key conditions, in the absence of a separate routine. The end-of-data and invalid-key routine address, when supplied, overrides the I/O error routine in the case of end-of-data and invalid-key conditions.

These routines are entered with the same addressability and protection status as for any other part of the user's program.

2.2 DOCUMENTATION

2.2.1 Naming Conventions

No name in the system may be over sixteen characters long. Thus no control block label may be over fifteen characters long (to allow for suffixing - see Appendix A). No external name (appearing in the Linkage Section of an object module) may be over eight characters long.

Uniform conventions for names of external entities (which may be referred to in other parts of the system) are useful in order to prevent conflicts of names as well as to make documentation more intelligible. The standards to be maintained by programmers of operating system routines are described in the following paragraphs.

2.2.1.1 External Names in Code

Names which appear as entry names in object text must be unique across the system. To insure this, a unique prefix for external names is assigned to each separately assembled or compiled portion of operating system text.

1. The first character of a name identifies WANG-supplied programs:

   W - WANG supplied

2. The second character of a name identifies the major area of system code to which it belongs:

   S - Central supervisor
   V - Supervisor call routines
   N - Initialization
   M - Command and HELP processing
   D - Data management
   P - System service tasks
   T - Operating system utilities
   U - Stand-alone utilities
   L - Linkage editors
   A - Assembler
   C - COBOL
   R - RPG
   B - BASIC
   E - Text editors
   I - Other independent components
   F - FORTRAN
   X - Data base
   W - Word Processing
   G - File management utilities

3. The third (and, for SVC routines, fourth) character identifies the specific functional area into which the code falls:

(S)	1	Task scheduler
	2	Clock interrupt handler
	3	I/O interrupt handler
	4	Program check interrupt handler
	5	SIO routine
	6	Pager
	7	Page, SVC, or I/O tracing
	8-end	Additional service subroutines
(V)	00-end	SVC routines by SVC number
(N)	1	System initialize (IPL)
	2	Task reinitialize routines
(M)	0-end	Command and HELP processing
(D)	0-end	Data management vectored routines and subroutines
(P)	1	Operator's console program
	2	File sharing handler
(U)	1	Disk volume analyze and label
	2	Display and alter disk
	3	Copy tape to disk
	4	Copy file, library, or volume
	5	Copy tape to disk (tape bootstrap)
	6-end	Assigned as needed
(T)	1	On-line disk volume initialize
	2	Display file
	3	Disk copy
	4	Tape copy
	5	Print listing file
	6	List Volume Table of Contents
	7	Sort
	8	Print debugging dump from diskette
	9	Copy unloaded IBM OS/370 PDS
	A	Patch file
	B	System procedure interpreter
	C	COPY2200
	D	FLOPYDUP
	E	BACKUP
	F	VERIFY
	G	TAPEINIT
	H	SECURITY
	I-end	Assigned as needed

(L) 1-end Linkage editors (binders)

(A) 1-end Assemblers

(C) 1-end COBOL compiler and COBOL symbolic debugging

(R) 1 RPG compiler

(E) 1 COBOL and Assembler Language source text editor

2-end Assigned as needed

(I) 1-end Other independent components

(G) 0 Common file management subroutines
1 CONTROL
2 DATENTRY
3 REPORT
4 INQUIRY
5 CONDENSE
6 KEYENTRY

(Programming in categories U, T, L, A, C, R, E and I would not normally contain entry names. These categories are included for completeness and to provide a noncompulsory standard for coding.)

4. Four or five characters uniquely identify the name. Some examples might be:

WS1SCHD
WS3STIM
WV04LINK
WD8WRIT

2.2.1.2 Names in System Data Structures

These are external symbols in the sense that more than one data structure description (control block definition) may be included in an assembly or compilation, and therefore, names must be unique across all these descriptions.

Each system data structure has a three-character to five-character name, as defined in Chapter 5 of this document. These names and any names formed by using these characters as a prefix are considered reserved names in assemblies of system code. Each label of a field within a system data structure definition must begin with these characters.

2.2.2 Macroinstruction Descriptions

Descriptions of all supervisor service macroinstructions, in alphabetic order by name, are collected in Chapter 4 of this document. Each description includes:

1. A functional title for the macroinstruction.

2. A syntax specification on the first line of the description, with optional operands displayed in brackets.

3. A specification of the restrictions placed on the use of the macroinstruction, such as 'for use by disk management support only' or 'register R1 must contain the active Task Control Block's address when this macroinstruction is executed', or 'for use by disabled routines only'.

4. A general functional description. This should describe what the macroinstruction accomplishes, without describing its internal workings in detail.

5. Descriptions of each operand and its effect on the function of the macroinstruction.

6. At least one example of code generated by this macroinstruction.

Examples of the format required may be found in Chapter 4.

2.2.3 Data Layouts and Descriptions

The format for describing system data structures is presented by example in Chapter 5 of this document. Note that the type of data in each named field of a data structure or control block is specified by the corresponding assembly language type designation and length (e.g., BL1, AL3, etc.), and that a brief description of the function of each field is included.

2.2.4 Supervisor Call (SVC) Routine Descriptions

Descriptions of all supervisor call routines are provided, in SVC number order, in Chapter 6 of this document. Each description includes:

1. SVC number and name.

2. A description of all direct and indirect inputs to the routine.

3. A description of all outputs (including 'side-effects').

4. A functional description of the processing performed by the routine.

CHAPTER 3: SYSTEM DESCRIPTION

3.1 VIRTUAL MEMORY

The user is not aware of the specific functions involved in providing him with a virtual main storage larger than the physical main storage of the hardware. However, an understanding of the mechanism involved will aid in understanding the protection mechanism and will be useful to the programmer who feels he must understand the Operating System's internals. Also, a general awareness of how paging is carried out may help any programmer write more efficient code for the machine.

3.1.1 Relation of Virtual Memory to Physical Memory

The physical main memory of the Wang 2200VS is limited to a maximum of 512K (K=1024) bytes. A computer instruction can, however, develop an address which refers to any of 16,777,216 (16,384K) memory locations. Thus a programmer could write a program which would execute in Wang 2200VS' physical memory only if a much larger memory were available. Being able to execute such a program means considerable savings in coding time, since the programmer then need not segment a large program into overlays. A similar situation arises with respect to the areas containing user-modifiable data. If the programmer must design for a system in which these areas are severely limited, he must make explicit use of secondary storage (disk, tape, etc.) to temporarily save intermediate results. This may result in more coding time, a more complex program, and considerable inefficiency.

The foregoing problems can be alleviated if the Operating System manages main memory in such a way that the user appears to have one or more large addressable areas of memory available. Under such a scheme, the Operating System overlays portions of the user's program and data with other portions as necessary, according to some rules which are sufficiently close to optimal to allow the program to proceed at nearly the speed at which it would execute if all of the program and all the work area for data were in main memory constantly. The memory which the user's

program addresses is then referred to as a 'virtual memory' and the actual main memory as the 'physical memory'. In the Wang 2200VS, one user's virtual memory includes a reentrant program segment of up to 512K bytes, a modifiable segment of up to 512K bytes, and a system code and data segment of up to 256K bytes. The virtual memory exists in complete form only on a secondary storage device (disk).

3.1.2 Address Translation, Pages and Page Faults

It is the relationship or 'mapping' between a large virtual memory and a smaller physical memory which the paging management routines of the Operating System provide. This is done as follows:

The physical memory is partitioned into 'page frames' of 2048 (2K) bytes each. (A 128K system would have 64 page frames.) Certain of these page frames are dedicated to contain Operating System routines and data which must be resident at all times. The other page frames may contain parts of any other programs or data. As far as an executing program is concerned, it may address any one of sixteen large contiguous areas of storage (up to 512 pages each) referred to as 'segments'. The particular segment addressed is selected by the four high-order bits of a 24-bit address. The user program code is in segment one, the user modifiable data area in segment two, and supervisory routines and data in segment zero. References to segments three through fifteen are invalid, and are treated as program errors. Actually, the hardware uses these four bits to select a 'page table', each item of which in essence addresses a page frame in the physical main memory. The nine bits of the original ('untranslated') address following the segment selection bits are used to select one of up to 256 page table items within a page table, and thereby to select a physical page. The last eleven bits of the original address are then used to address a byte location within this page. The resulting addressed location is referred to as the 'translated' address. Note that although up to 256 page table items may be addressed within a page table, there is a maximum of 64 page frames available in the physical memory of the 128K system hypothesized above. Some page table items are therefore marked with a special indication that the referenced page is missing from physical memory. The hardware, finding this indication set during an address 'translation', effects a particular type of program check interruption known as a 'page fault', and supplies to the program interrupt service routine the segment and page numbers of the missing page. A task initiated by this interrupt service routine, the 'paging task', attempts to locate a page frame the contents of which may be replaced with the required page from a disk file. When a page frame containing a replaceable page has been selected, the paging task reserves this page by indicating that it is in use for page-in or page-out, and initiates the page-in, or page-out followed by page-in, operations. The task which needs the page is forced to wait on a queue of tasks attached to the page frame

being used for the paging operation, and will be reactivated after the page-in operation completes and the paging task has updated the page table to allow normal addressability of the page.

3.1.3 User Program Efficiency and Paging

Designing a program for a virtual memory environment requires a new outlook toward program and data organization. Although the user is freed from the onerous task of managing a small physical storage by overlay or other manual segmentation techniques, he cannot ignore the issue of program organization. A major aim of the programmer should be to increase the locality of reference of his program's code. That is, he should avoid referencing many pages of code or data within a short span of program execution. In this way he will reduce the likelihood of many page faults occurring. One might also say, as a general rule, that it pays to spend processor time to save space; the notion of a tradeoff between the amount of storage area required and the speed at which a program executes is not, however, as accurate in a virtual memory system as in most previous systems.

3.2 SUPERVISORY FUNCTIONS

The following paragraphs describe briefly the most important functions of the central supervisor. System programmers requiring more detailed information should consult program logic documents.

3.2.1 Task Scheduling and Timing

The operating system supports one or more tasks corresponding to users' programs in progress, in addition to a paging task. Determination of which task is to have control of the processor at any time is made by the scheduler routine, which gains control:

(1) when a task WAITs on a semaphore (associated with an event which is expected to occur but has not yet occurred).

(2) when a task becomes ready as a result of a SEND on a semaphore. (This is associated with the occurrence of an event.)

(3) when an interrupt has occurred, and preliminary processing of the interrupt has completed (e.g., on exit from the I/O interrupt service routine or clock interrupt service routine).

(4) when 'System Must Complete' state is exited, and a workstation HELP key had been received while in 'System Must Complete' state.

The scheduler determines the highest-priority ready task, sets a timer expiration value to insure that this task cannot monopolize use of the processor at the expense of other tasks (most obviously by looping), and loads the control registers, page tables, general registers, floating-point registers, and the program control word (PCW) associated with the task to be given control. This task is allowed to run until one of (1) through (4) above occurs.

3.2.2 WAIT and SEND Primitives

When a task is not available for scheduling, it is said to be 'blocked'. Blocking and unblocking of tasks is accomplished by the WAIT and SEND primitives which execute DSEM or ISEM instructions, respectively. The functions are as follows:

(1) WAIT - This function is always performed by a special operating system WAIT routine. An addressed semaphore's count field is decremented. If the result is zero or greater, control returns to the invoking program. If the result is less than zero, the currently active (invoking) task is blocked and control is passed to the scheduler. User programs must use the CHECK SVC routine to gain access to this facility.

(2) Multiple WAIT - This function is similar in effect to WAIT, but allows a task to await the occurrence of one-out-of-several specified events, each represented by a semaphore.

(3) SEND - This function is performed by a special operating system SEND routine as requested by an SVC call or special branch-type entry for interrupt routines. The SEND routine increments an addressed semaphore's count field (by executing an ISEM instruction). If the result is less than or equal to zero, a task has been unblocked by this instruction, and the routine passes control to the scheduler. If the result is greater than zero (no task unblocked), control is simply returned to the invoker. (Interrupt routines either pass control to the scheduler or to the interrupted task at some time after a SEND branch type entry.)

Users' programs do not have the ability to modify semaphores except by invoking the CHECK routine through supervisor call. After checking the request for validity, CHECK enters the WAIT routine. All semaphores are in protected memory locations.

3.2.3 Intertask Message Primitives

The XMIT primitive allows a task to queue a message of up to 251 bytes for transmission to another task. The WAIT primitive (through the CHECK MESSAGE macroinstruction) allows a task to wait for receipt of such a message.

3.2.4 Supervisor Call Interruptions

Section 2.1.2 above describes the functions performed by SVC routines. The reader may wish to refer to this section and to Figure 2-3.

3.2.5 I/O Initiation

Issuance of the actual SIO instruction to initiate either paging or normal I/O is localized in the system's Start I/O routine. No other system components issue SIO instructions. The Start I/O routine is passed a Request Element which describes the I/O operation to be performed and points to a semaphore on which the I/O interrupt service routine will perform a SEND when the operation is complete. The Start I/O routine is called by the XIO supervisor call routine and the paging task. The Start I/O routine itself queues the Request Element to the proper device's physical request queue, and then attempts to address the first element on this queue and perform its requested I/O operation. If the device is busy, this step is bypassed (I/O will be started later, when the current I/O completes).

3.2.6 I/O Interruptions

The I/O interrupt service routine gains control on an I/O interruption. As well as 'unfixing' the I/O area (removing the page or pages it occupies from 'temporarily resident in memory' status) on I/O completions other than page faults, and saving all necessary status of the interrupted task, it locates the I/O Request Element associated with the operation which has completed, performs a SEND operation on the semaphore addressed by this element, and dequeues the element. It then calls the system's Start I/O routine to initiate I/O for any requests which may be pending. Exit is then made to the interrupted task, or to the scheduler to restore the status of another task and dispatch it. Interruptions indicating 'I/O processor now ready' cause the I/O interrupt service routine to attempt to initiate operations on all affected devices (those on the same I/O processor) which have request elements queued for servicing. 'Attention' interruptions are also processed by the I/O interrupt service routine, which performs special device-dependent processing of 'attention' interruptions from workstations.

3.2.7 Clock Interruptions

The system's clock interrupt handler receives control when a clock interruption occurs. If a task is active and the interruption indicates the end of its time slice, it is marked 'timeslice expired', placed last on the low priority scheduling queue, and the scheduler is entered to schedule another task, or the same task again if no other is ready. If the interruption signals the expiration of a program-specified timing interval, the clock interrupt handler performs a SEND operation on the semaphore in the appropriate interval timing element (TQEL), and then exits through the scheduler to schedule a task which may have been unblocked by this SEND operation. If the interruption signals a midnight crossing, the system date is updated and the time-of-day reset.

3.2.8 Program Interruptions (Other Than Page Faults)

The program interrupt handler entered on a program interruption will force entry to a program exception exit routine if one has been specified for this type of exception, and otherwise to the system's HELP processor. This will allow the user to request use of a debugging routine, or to simply terminate the program.

3.2.9 Program Normal Termination

Normal termination processing is entered on RETURN from a program originally entered from the command processor, or on execution of an UNLINK supervisor call by such a program. The memory occupied by the program is made available for other use. Open files are closed and any logging information is written.

3.2.10 Program Abnormal Termination

Abnormal termination (processing "cancellation") may occur in response to one of the following:

(1) The user requested abnormal termination in response to Help Processor prompting, the user having obtained the services of the Help Processor by striking the workstation's HELP Key.

(2) The program issued a CANCEL supervisor call or entered the Debug Processor by virtue of a program check (without having issued a PCEXIT supervisor call) and the user requested abnormal termination in response to Debug Processor prompting.

(3) The program issued a CANCEL supervisor call or program checked (without having issued a PCEXIT supervisor call) and the program itself had previously issued a CEXIT supervisor call with the NODEBUG or DUMP option.

The abnormal termination processor (Cancel Command Processor) will attempt to close files open to the program being terminated. Assuming that the program has not issued a CEXIT supervisor call specifying an address of a user "cancel exit routine," the program is removed from execution and control is passed to the command processor to allow another program to be initiated.

Programmed interception and analysis of abnormal termination conditions is available to the user program via the CEXIT supervisor call. The "cancel exit" parameter of the supervisor call specifies the address of a routine in the user's program that is to receive control in the event of an abnormal termination condition.

Upon entry to a user program's "cancel exit routine," the program name, PCW, general registers, and cancellation codes and message are provided on the system stack for subsequent analysis. The floating point registers reflect their value at the time of the abnormal termination condition. The program may take any desired corrective action and subsequently elect to either continue processing normally or terminate. (See "CEXIT Supervisor Call" for additional details.)

3.2.11 Program Initiation

The initiation of user programs is a function of the system's Command Processor and Procedure Interpreter, which issue LINK SVCs to initiate programs.

3.2.12 Microcode Loading

When an IOSW is received that has bits PP (peripheral processor code missing) or DP (device code missing) set, the I/O interrupt routine marks the UCB 'not busy' (UCBSTATBUSY reset) and 'no code' (UCBSTATNOCODE set), increments the I/O completion semaphore, and leaves the IORE on the I/O queue with the pages used for the I/O operation fixed. IOREREQFIVRQ is set to simulate an IVRQ interrupt. SIOs are blocked by UCBSTATNOCODE.

The task waiting on the I/O is restarted, and enters CHECK completion code. CHECK finds IOREREQFIVRQ set and enters 'intervention-required' processing. It then finds PP or DP set in the IOSW, and calls SVC LOADCODE to load the appropriate microcode. If LOADCODE is successful, it will have marked the UCB 'code loaded' (UCBSTATNOCODE reset), and restarted the I/O operation. CHECK then waits for I/O completion.

If LOADCODE failed, CHECK calls HALTIO to remove the I/O operation from the I/O queue and unfix the data area. CHECK then reports the IOSW as an error completion; if the device is a Workstation, it marks the device 'turned off' (UCBSTATNOTOP set) to block I/O operations until the Workstation is turned on. When the Workstation is turned on , the unsolicited interrupt will mark the device 'turned on' (UCBSTATNOTOP reset) and 'loaded' (UCBSTATNOCODE reset) to unblock I/O operations. The next I/O operation to the device will find the microcode not loaded and reload it.

Receiving an I/O completion interrupt with DP and PP both set may indicate that the device configuration table is missing. This is handled like an interrupt with 'DP' or 'PP' set.

3.2.13 Logoff

Logoff is defined as the termination of a user's current session with the computer. It may be invoked in either of the following manners:

(1) The user may request Logoff voluntarily by keying the appropriate initiator function key. In this particular case, the Logoff Processor permits the user to "escape" from Logoff by pressing the HELP key.

(2) The program may issue a LOGOFF supervisor call in response either to an interactive user request or any other specified conditions detected by programming. Such a logoff is essentially handled as a program cancellation, without entry to the Debug Processor, unlinking all link levels, bypassing the Initiator, logging off the current user, and displaying the Logon Processor screen on the workstation. A message is displayed indicating that the previous program was logged off by "program request." Note: LOGOFF supervisor calls may be intercepted by a program's active "cancel exit routine," distinguished by message 0001 issued by SVC43.

The Logoff Processor, in both the above cases, prompts the user for a response for each volume mounted by that user for exclusive use. The user is provided with a choice of changing the mount status from "exclusive" to "shared" or, where appropriate, dismounting the volume.

3.3 DATA MANAGEMENT FUNCTIONS

A general introduction to the implementation of Wang 2200VS data management follows. The system primitive functions provided to aid in processing I/O requests are explained briefly in Section 3.3.2. For more detailed information, consult the appropriate program logic documents.

3.3.1 Top Level Data Management Function Support

Entry to system code for processing data management requests is by indirect branch to routine addresses in a file's User File Block. Most data management code is executed in unprivileged mode as an extension of a user's program.

3.3.2 Data Management Support SVC's

Primitive I/O services are provided through supervisor call routines which are to be used by the data management system in the process of servicing READ, REWRITE, WRITE, DELETE, START and other requests. The XIO and CHECK services may also be used in nonstandard I/O programming.

3.3.2.1 XIO

The XIO service provides intermediate-level I/O initiation support. When called by SVC from data management programs, it converts block numbers in a disk file to disk addresses, validates that disk addresses lie within proper extents, translates virtual data addresses (in I/O Command Words) to physical addresses, makes the required pages of virtual memory temporarily resident ('short-term fixed'), and calls the supervisor's Start I/O routine to initiate the I/O operation.

3.3.2.2 ALEX

The ALEX service is requested by data management routines when a primary or secondary extent on disk is exhausted and it is necessary to either allocate another extent or, failing that, to terminate processing.

3.3.2.3 CHECK

The CHECK service is used by data management routines to suspend the operation of the issuing task until an I/O operation is signalled complete (by a SEND function in the I/O interrupt service routine). The user-level data management routines must use CHECK, since they are not allowed direct use of the WAIT primitive function. Basically, the difference between these functions is that CHECK will insure that a valid Open File Block address has been passed to it, while WAIT cannot validate the semaphore address which it receives. CHECK will also handle I/O error logging and detection of 'intervention required' conditions at the device. (See the CHECK supervisor call and macroinstruction descriptions for additional functions of CHECK, including its role in interval timing support and inter-task message passing.)

3.4 DISK STORAGE DESCRIPTION

Disk volumes on the Wang 2200VS system are divided into logical 256-byte sectors, numbered from zero. Actual disk sectorization is into 256-to-2048-byte sectors, depending on the device type. Although each actual sector may be addressed by I/O command, the operating system XIO routine and paging routines always address 2048-byte areas referred to here as 'blocks' of disk storage, which begin on logical sector addresses which are multiples of eight. Files on such a volume are recorded in one or more contiguous areas ('extents'). Each extent spans one or more consecutively-numbered blocks. The presence of a file is indicated in a volume table of contents which is organized to allow a hierarchical naming scheme with one level of qualification. This structure is located through the volume label. All these items are discussed in the following paragraphs.

3.4.1 Volume Label

The volume label occupies sector 1 (the second sector) of any disk volume. It contains the name of the volume (volume serial number), the location (extent descriptions) of the volume's table of contents, and other descriptive information defining the size and physical organization of the volume.

3.4.2 Extent Organization

Each block on a volume, with the exception of the first block (sectors 0 to 7) and the blocks containing the volume table of contents, is part of an 'extent' (defined contiguous area) of either free space or file space. Extents of free space are recorded in available space records of the volume table of contents. Extents of file space are recorded in File Descriptor Records (FDRs) in the volume table of contents, where each FDR is associated with a particular file. The initial FDRs for files are referred to as Type 1 FDRs. FDRs describing additional areas occupied by a file are referred to as Type 2 FDRs.

When a file is initially allocated space, an attempt is made to acquire a single extent of sufficient size on the specified volume. If such an extent is not available, up to 3 extents may be allocated.

When a file is enlarged so that it exceeds the capacity of extents previously allocated for it, the system allocates an additional extent. This may require that an additional FDR be allocated to contain the additional extent information (a Type 2 FDR). A file may encompass a maximum of 13 extents (described in one FDR1 record and one FDR2 record).

3.4.3 Volume Table of Contents

The volume table of contents on a disk volume has blocks (8-sector areas) of four types. There are available space blocks which record where free blocks are on the volume, high-level index blocks which contain file name qualifiers and low-level index block numbers, low-level index blocks which contain file names and file descriptor block numbers and file descriptor blocks containing File Descriptor Records which contain attribute specifications for particular files, including the location of the file on the volume and the extents allocated for it.

The first block of the volume table of contents is an available space block. Any search for available space on the volume (for a newly-allocated file or an additional extent) begins by searching this block, followed by its chained blocks, for a sufficiently large area of space.

The second block of the volume table of contents is an index block. It is the first block of the 'top-level' index. Additional top-level index blocks may be chained from this block. This top-level index contains pointers to lower-level index blocks, which in turn contain pointers to File Descriptor Records. The name of a lower-level index, as recorded in each of its index records, is used as a file name qualifier to define a complete file name or 'path name'. For example 'DIRECTB.FILEA' could be the name of a file whose File Descriptor Record is located by a pointer in an index item containing 'FILEA', where this item is in a lower-level index record whose name is 'DIRECTB'. It is useful to refer to the file as member 'FILEA' in library 'DIRECTB'. The first high-level index block also contains three linked-list head pointers, which are used to maintain chains of

(1) available blocks,
(2) blocks containing available low-level index records, and
(3) blocks containing available File Descriptor Record areas.

The third block of the volume table of contents is reserved for use as the first lower-level index block. Each such block may contain all or part of up to four 'libraries' as described above.

The fourth block of the volume table of contents is a block containing File Descriptor Records.

The fifth and additional blocks of the volume table of contents are used as required for available space blocks, index blocks, or file descriptor blocks.

CHAPTER 4: SYSTEM MACROINSTRUCTIONS

4.1 MACROINSTRUCTIONS AVAILABLE

This chapter documents the Wang-defined macroinstructions available for general programming use. A second set of macroinstructions, designed for operating system development and support and not intended for general use, are documented in a separate manual.

The available macroinstructions are summarized in the following list, arranged by functional category. The remainder of this chapter consists of individual discussions of all general-purpose macros, arranged alphabetically by keyword.

A. User program linkage

CALL, RETURN, LINK

B1. User-level I/O

OPEN, CLOSE, READ, REWRITE, WRITE, DELETE, START, RENAME, SCRATCH, UFBGEN, BCTGEN, AXDGEN, MOUNT, DISMOUNT, READFDR, TCOPTION

B2. Data management routine use

XIO, GETBUF, FREEBUF, CHECK, ALEX, HALTIO

C. Synchronization and resource control

WAIT, SEND, SEIZE/RELEASE (Not available for general programming use)

D. Resident block management

GETMEM, FREEMEM (Not available for general programming use)

E. Timing

TIME, SETIME, RESETIME, CHECK

F. Program termination and debugging

CANCEL, (RETURN)

G. Workstation display, message log

GETPARM, KEYLIST, MSGLIST, FMTLIST, PUTPARM

H. Program structuring and control

REGS, EXTRACT, PCEXIT, SET
LOW (Not available for general programming use)

I. Intertask communication

XMIT, CHECK, CREATE, DESTROY

Allocate Extent

Syntax

```
[label]  ALEX  OFB=(register)
```

Restrictions

For use by Data Management System routines only.

Function

Attempts to allocate an additional extent for the disk file whose Open File Block (OFB) is addressed. A file has to be opened for exclusive use before ALEX is issued. The OFB address is stacked before issuing the Supervisor Call. On completion of the function, the OFB address is removed from the stack and a return code word is stacked, as follows:

- 0 - Additional extent allocated
- 4 - Invalid OFB address or OFB not open for this task; no allocation
- 8 - File in INPUT mode; no allocation
- 12 - Wrong device class; no allocation
- 16 - Extent limit would be exceeded; no allocation
- 20 - All buffers in use, GETMEM failure; no allocation
- 24 - Volume full; no allocation
- 28 - No space in VTOC for FDR2; no allocation
- 32 - Disk I/O error; VTOC unreliable

Operand Description

OFB= The address of an Open File Block for an open disk file. It must be presented as a register specification in parentheses, where the register is assumed to contain the OFB address.

Example

```
 LAB1     ALEX     OFB=(R1)
+LAB1     PUSH     0,R1
+         SVC      14 (ALEX)
```

Generate Alternate Index Descriptor Block

Syntax

```
[label]  AXDGEN  [MASKSIZE=p1] [,ENTRIES=p2]
                 [,(ORD=p3, KEYPOS=p4,
                    KEYSIZE=p5 [,NODUPS]
                    [,COMPRESS])] ....
```

Function

Generates an alternate index descriptor block (AXD1) to be addressed by UFB field UFBALTPTR (ALTAREA operand of UFBGEN macroinstruction).

Operand Descriptions

MASKSIZE = Must be 2 or omitted.

ENTRIES = To use the AXD1 for OUTPUT mode processing, must equal the number of positional operands (in parentheses) which follow. Always must be zero, omitted, equal to the number of positional operands, or 0 to 16 if there are no positional operands.

Positional suboperands:

ORD = Index (1 to 16) defining this alternate index structure (access path). Required in all supplied positional operands.

KEYPOS = Key position in record. Required.

KEYSIZE = Key length. Required.

NODUPS Duplicates not allowed if specified for OUTPUT mode. Ignored in other modes.

COMPRESS Key compression if specified for OUTPUT mode. Ignored in first version of alternate indexing support.

Example:

```
 LAB1     AXDGEN ENTRIES=1
+LAB1     DC     F'0'              BL
          DC     XL14'0'           MASK, UFB, ALTINX, FLAGS
          DC     HL1'2'            MSIZE
          DC     XL41'0'           SPARE1, BCB, PMASK, SPARE

 *AXD ENTRY FOR ALTERNATE ACCESS PATH
          DC     AL1(0)            XORD
          DC     BL1('10000000')   FLAGS
          DC     H'0'              LEVELS
          DC     AL2(0)            KEYPOS
          DC     AL1(0)            KEYSIZE
          DC     XL21'0'           HXBLK, NRECS, PTRD, ESPARE
```

Generate a Buffer Pool Control Table

Syntax

```
[label] BCTGEN NBUF=expression
```

Function

Generates a skeleton buffer pool control table for use in buffer pooling (UFBGEN macroinstruction, operands POOL and BCT).

Operand Description

NBUF= An absolute expression must be supplied, which must evaluate to an integer not greater than 60. This is the number of buffers to be included in the buffer pool.

Example

```
 LAB1      BCTGEN     NBUF=8
+LAB1      DS         0F
+          DC         AL1(8)
+          DC         XL19'00'
+          DC         (8)XL28'00'
```

Call a Subroutine

Syntax

```
                                                [                              ]
[label]  CALL  EPLOC=Entry-address-word         |,PARM={(register)}            |
                                                |      { label    }            |
                                                |                              |
                                                | PARMLOC=parm-address-word    |
                                                [                              ]

          [,COND=number]
```

Restrictions

A stack, with stack top addressed by general register 15, must be available to the CALLer.

Function

Loads the address of a parameter list (if specified) into register R1 and branches (conditionally) to the specified label or to the address contained in 'entry-address-word' by a JSCI instruction (leaving the return address on the stack). The JSCI instruction saves the contents of control register 1. The JSCI instruction also stores general registers 0 to 14 on the stack and places the address of the register 0 save area in control register 1 (as well as in the stack pointer, general register 15). The lowest address in any current 'static' area is (by convention) passed in register R14.

Operand Descriptions

EPLOC= The address of a word containing the CALLed routine's entry point address. This must be specified in a form allowable in the D2(X2,B2) fields of the RX-type assembler instruction format.

PARM= The address to be passed in register R1.

PARMLOC= The address of a word containing the address to be passed in R1 (with format as for EPLOC).

COND= Specifies the condition codes under which the routine is to be CALLed. If omitted, 'COND=15' is assumed.

Example

```
 LAB1     CALL    PARM=PADDR,EPLOC=ENTRYWRD,COND=8
+LAB1     LA      R1,PADDR
+         JSCI    8,ENTRYWRD
```

Cancel

Syntax

```
[label]  CANCEL MSG={(register)}
                    { address  }
```

Restrictions

Must not be issued while in System Must Complete state.

Function

To enter the Help Processor for cancellation of the issuing program. The message provided is displayed, along with a standard CANCEL message. The user may then use the Help Processor's debugging facilities to examine the program before issuing a CANCEL command to remove it from the system. If the CANCEL supervisor call was issued from within a user's program, the user may attempt to continue processing by modifying the location to receive control and invoking the "CONTINUE" command. A program terminated by a CANCEL supervisor call from within privileged code cannot be continued.

Operand Description

MSG= The address of a message to be displayed, contained in the specified register, or at the specified address. A register specification must be parenthesized as shown. The message must be in the format generated by the MSGLIST macroinstruction.

Example

```
          LA        R5,LAB2
 LAB1     CANCEL    MSG=(R5)
+LAB1     PUSH      0,R5
+         SVC       16 (CANCEL)
           .
           .
           .
 LAB2     MSGLIST   'C001','SUPVSR','MEMORY POOL
                    EXHAUSTED'
+LAB2     DC        CL4'C001'
+         DC        CL6'SUPVSR'
+         DC        AL2(21)
+         DC        C'MEMORY POOL EXHAUSTED'
```

Cancel Exit

Syntax

```
(1) [label]   CEXIT  CANCEL

(2) [label]   CEXIT  SET [,([NODEBUG][,NOHELP])]
                         [  [DUMP   ]          ]

                          [,ADDRESS={(register)}]
                          [         { expression}]

                          [,MESSAGE={(register)}]
                          [         { expression}]
```

Function

CANCEL - Negates the effect of any previously issued CEXIT supervisor call in the current link level. Abnormal termination (cancel) conditions will not be intercepted at the current link level.

SET - To specify user-program abnormal condition handling for the current and any subsequent link levels. The options specified may be negated via the CANCEL option or reset via another SET option at the current link level, or may be temporarily overridden at subsequent link levels via the SET option issued therein.

Operand Descriptions

NODEBUG - The Debug Processor is bypassed for abnormal termination conditions. Control is passed directly to the Cancel Command Processor without direct user notification.

DUMP - Similar to the NODEBUG option, this option also provides a full program dump prior to entry into the Cancel Command Processor.

NOHELP - Causes HELP key to be disabled at current link level for the purposes of entry into the HELP Processor. If NOHELP is specified, pressing the HELP key in user mode has the following effects: 1) If the workstation does not have operator privileges, the alarm is sounded; 2) If the workstation is a dual mode operator console, operator mode is entered. This option remains in effect until a CEXIT without the NOHELP option is issued or until the program unlinks back to either the Command Processor's Initiator or a link level for which NOHELP was not specified. Unless specifically disabled

therein, the NOHELP option is propagated to higher link levels. The NOHELP option should only be utilized in situations in which user access to CANCEL and other system functions must be limited, such as in the case of critical sections of application programs updating multiple file chains and pointers. Such programs should be highly debugged prior to use of this facility.

ADDRESS= - Specifies the address of a user-program provided cancellation intercept routine. This routine gains control from the Cancel Command Processor in the following manner:

If the abnormal termination condition occurred within the same link level, the Cancel Command Processor returns control to the program at the address of the cancellation intercept routine. (This routine may also gain control if the current or any subsequent link level issues a LOGOFF SVC and this CEXIT option is still active.) The registers are those at the time of either the program check or entrance to the supervisor call resulting in the abnormal termination condition. Thus, addressability should not be assumed. If the abnormal condition occurred while Data Management for either disk or tape was in control, an attempt is made to complete that operation. All non-I/O wait conditions are removed. The Cancel Command Processor does not, in this case, attempt to close any files.

If the abnormal termination condition occurred within a subsequent link level, i.e., a "Linked-to" program, for which no cancellation interception routine was specified, the Cancel Command Processor successively attempts to complete I/O operations and close files and then UNLINK each link level until a link level with a cancellation interception routine (if any) is found, at which point control is passed to that routine. In this case, the registers are those at entry to the LINK supervisor call. As in previous cases, all non-I/O wait conditions are removed.

In both cases, entry to the cancellation intercept routine cancels the CEXIT options for the link level. They may, if desired, be reset via a subsequent CEXIT supervisor call. On the stack, the cancellation intercept routine finds the following data (which may be accessed symbolically via the DSECT produced by the CXT macroinstruction):

CANCELLATION PCW	-	8 Bytes
PROGRAM NAME	-	8 Bytes
(Reserved)	-	48 Bytes
GENERAL REGISTERS 0-15	-	64 Bytes
CANCEL MSGLIST	-	Variable Length

MESSAGE = Provides text to be used by both the Help Processor and the Debug Processor in place of the "CANCEL PROCESSING" menu descriptions. Specification is of a segment 2 location containing a one byte binary length field followed by up to 27 bytes of text. Specification of this option is independent of any user cancellation intercept routine specification.

Example

```
 LAB1        CEXIT  SET,NODEBUG,ADDRESS=FIXPROBS,MESSAGE=CANCELME
+LAB2        PUSHA  0,0
+            MVI    0(15),B'00000000'
+            MVI    1(15),B'00000000'
+            PUSHA  0,CANCELME
+            PUSHA  0,FIXPROBS
+            MVI    0(15),B'10100000'OPTIONS BYTE
+            SVC    39 (CEXIT)
```

Check For Event Occurrence

Syntax

```
(1) [label]  CHECK    {OFB} = {address   } [,ERREXIT= {address   }]
                      {VCB}   {(register)} [          {(register)}]

                    [,IOSWREG=R0]

(2) [label]  CHECK    INTERVAL

(3) [label]  CHECK    MESSAGE= {address   }     ,PORT= {address   }
                               {(register)}            {(register)}
                                                       {'string'  }

(4) [label]  CHECK    WSKEY  = {address   }
                               {(register)}

(5) [label]  CHECK    INTERRUPT= {address   } ,IOSWADDR= {address   }
                                 {(register)}            {(register)}

(6) [label]  CHECK    TCIO, OFB= {address   }
                                 {(register)}

                    [,IOSWADDR= {address   }]
                    [           {register  }]

(7) [label]  CHECK MULTIPLE,PLIST= {address   }
                                   {(register)}

                         ,COUNT= {self-defining term}
                                 {(register)        }
```

Restrictions

CHECK OFB or VCB should be issued only after issuing an XIO call.

Function

(1) CHECK OFB or VCB waits for completion of an I/O operation. If 'intervention required' is indicated on completion, CHECK issues an appropriate workstation message (if possible) to inform the user, and proceeds when the 'intervention required' condition has been cleared. (CHECK may reissue the message if the condition has not actually been corrected.) CHECK waits for completion again after the condition has been cleared. If the operation has not completed, CHECK suspends processing of the issuing program until completion. If the ERREXIT operand is provided, CHECK returns to the specified address in the event of a permanent error completion (IOSW bit EC set, bits NC or IRQ not set).

Otherwise CHECK returns to the next sequential instruction address. CHECK logs I/O errors by means of a nonresident subroutine.

(2) CHECK INTERVAL waits for expiration of a timing interval as set by the SETIME macroinstruction.

(3) CHECK MESSAGE waits for a message to be sent to the issuing task.

(4) CHECK WSKEY waits for a Program Function Key to be struck on the specified workstation, which must be reserved for use by the issuing task. An un-CHECKed XIO request must not be outstanding to this workstation when this CHECK is issued.

(5) CHECK INTERRUPT waits for an unsolicited interrupt from a workstation, a printer, or a telecommunications device.

(6) CHECK TCIO waits for the occurrence of a TC I/O event. This event may be a completion of an I/O operation which was previously initiated by a call of the XIO SVC by the RECEIVE or TRANSMIT macro. This event may also be an unsolicited interrupt from a Data Link Processor (DLP) if no previous I/O command was issued.

(7) CHECK MULTIPLE waits for any one of several specified events to occur. These can be any of (1) through (6) above. For details of parameter list construction, refer to the CHECK SVC description (SVC 17).

NOTE: A FORM=LIST operand may be used with functions 1-6 above to build a multiple CHECK list on the stack. For example,

```
CHECK INTERVAL, FORM=LIST
CHECK WSKEY=(R1), FORM=LIST
LR    R9,SP
CHECK MULTIPLE, PLIST=(R9),COUNT=2
```

will build a multiple CHECK list which waits on a PF key or a timer, in that order. After the call to CHECK MULTIPLE, the top stack word will contain the offset into the parameter list of the event that occurred. The parameter list remains on the stack.

Operand Descriptions

OFB= For the OFB option, the address of the Open File Block (OFB) for a file previously OPENed. Must be presented as an address expression, or as a register specification in parentheses where the register contains the address of the OFB.

For the TCIO option, OFB= points to the address of the Open File Block (OFB) for the I/O channel device used in the I/O operation initiated by the corresponding RECEIVE or TRANSMIT call. The address supplied in the OFB= operand is an address pointing to a four-byte field containing the address of the OFB in the low-order three bytes.

VCB= The address of a Volume Control Block (VCB). May be used only if the caller is in System Mutual Exclusion (SME) or the volume is mounted for initialization. Must be presented as an address expression, or as a register specification in parentheses where the register contains the address of the VCB. Note that the displacement constant of +1 is added to the VCB address by the macroinstruction code in order to distinguish the CHECK VCB option from the CHECK OFB option.

ERREXIT= Optional address of an instruction to receive control in the event of an I/O error. Must be presented as an address expression, or as a register specification in parentheses where the register contains the error exit address.

IOSWREG= If IOSWREG=R0 is specified, the completion IOSW will be returned in general registers 0 and 1.

MESSAGE= An address in segment 2, into which a received message will be placed. The receipt area in segment 2 must contain the total length of the area, in binary, in its first two bytes. The length must not be greater than 2016 bytes. The message is placed in the specified area. If the area length is less than the message length plus two, the message will be truncated on the right. The area length bytes are updated to reflect the length of the message, plus two. (This is the full length of the message, even if the message was truncated.)

PORT= The four-character name of one of this task's active message receipt ports, as established by CREATE. May be specified as an expression addressing a 4-byte field containing the port name, as a register in parentheses pointing to the 4-byte field containing the port name, or as a character string in single quotes which is the port name.

WSKEY= A workstation device number. Specified in the low-order byte of the 4-byte field pointed to by an address expression, or in the low-order byte of a register in parentheses.

INTERRUPT= The device number of a workstation, printer, or telecommunications device. May be specified in the low-order byte of the 4-byte field pointed to by an address expression, or in the low-order byte of a register in parentheses.

IOSWADDR= An address in segment 2, into which the IOSW will be placed. May be specified as an address expression, or as a register in parentheses containing the address of the IOSW receipt area. This operand is required for the CHECK INTERRUPT option and for the CHECK TCIO option if the CHECK is for a TC unsolicited interrupt.

The IOSWADDR operand is not required for the TCIO option if CHECKing for completion of an TC I/O event.

PLIST= Address of a parameter list for CHECK MULTIPLE. May be specified as an address expression, or as a register in parentheses containing the address of the parameter list.

COUNT= Number of events (PLIST entries) for CHECK MULTIPLE. May be specified as a self-defining term which is the number of events, or as a register in parentheses containing the number of events (in binary) in the low-order byte.

Example

```
LAB1    CHECK   OFB=(R2),ERREXIT=ERROR
+LAB1   PUSHA   0,ERROR
+       PUSH    0,R2
+       SVC     17    (CHECK)
```

Close File

Syntax

```
[label]  CLOSE  [REEL     ,] UFB={(register)}
                [NOREWIND,]     {expression}
                [UNLOAD   ,]
```

Restrictions

None.

Function

Closes a file (removes it from processable status). Places the User File Block (UFB) in a state in which an OPEN can be addressed to it to return the file to processable status. This includes placing sufficient file location information in the UFB so that a succeeding OPEN will refer to the same file, volume, and device. If UFB bit UFBF1WORK is set and the file is in a library named #xxxWORK (where xxx is the USERID), the file will be SCRATCHed as well as CLOSEd. (See the SCRATCH supervisor call for function.)

Operand Descriptions

UFB= The address of a User File Block for an open file. It must be presented as a register specification in parentheses, where the register is assumed to contain the UFB address, or as a UFB address expression not in parentheses. If omitted, only the SVC instruction is generated.

REEL If specified for a file on an appropriate device (e.g., magnetic tape), the file will not be closed, but rather will be positioned so that the first record on the next volume (if any) will be provided on the next READ, or written on the next WRITE.

NOREWIND If specified for a magnetic tape file, rewinding is suppressed when the file is closed.

UNLOAD If specified for a magnetic tape file, the tape will be rewound, set to "offline," and effectively DISMOUNTed when the file is closed.

Example

```
 LAB1     CLOSE    UFB=(R3)
+         PUSH     0,R3
+         SVC      1 (CLOSE)
```

Create Intertask Message Port

Syntax

```
[label] CREATE PORT= {(register)} ,BUFSIZE= {(register)}
                     {expression}           {expression}
                     {'string' }

                     [,PRIVILEGED]
```

Function

Activates an intertask message receipt port with the specified port name, and with the issuing task as the valid receiver. Optionally screens out messages not transmitted by code in privileged state or dedicated system tasks.

Return codes are placed in the word on the stack top as follows:

- 0 - Successful.
- 4 - Another task has activated the specified port name.
- 8 - Same task has already activated the specified port name.
- 12 - GETMEM failure.

Operand Descriptions

PORT= The four-character name of a message receipt port (chosen by the issuing program; any characters are allowed). May be specified as a register in parentheses pointing to the port name, as a literal in single quotes which is the port name, or as an expression addressing a 4-byte field containing the port name.

BUFSIZE= The space in bytes to be allocated for buffering messages. May not be greater than 2016.

PRIVILEGED Causes only messages transmitted by tasks in privileged code or by dedicated system tasks to be received by the message receipt port being created.

Example

```
 LAB1   CREATE  PORT=PORTNAME,BUFSIZE=(R0)
+LAB1   PUSHC   0(4,0),PORTNAME
+       PUSH    0,R0
+       MVI     0(15),X'00'
+       SVC     37 (CREATE)
```

Return CEXIT 'RETURN' Information

Syntax

```
CXT   [NODSECT] [,REG=expression] [,SUFFIX=character]
```

Function

The CXT macroinstruction allows the user to symbolically reference the information returned to a program's cancellation interception routine.

Operand Descriptions

NODSECT - Specification of NODSECT results in the CXT fields to be assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, a DSECT with the name CXT (+ SUFFIX) is generated.

REG= - Provides for the optional specification of a register for which a USING statement for the CXT fields is generated.

SUFFIX= - If provided, all labels are generated by the concatenation of 'CXT', the user-provided SUFFIX (one ASCII character in length), and the field name.

Example

```
            CXT
+CXT        DSECT
+*          THE CEXIT RETURN INFORMATION BLOCK IS RETURNED TO
+*          A PROGRAM'S CEXIT ROUTINE FOR PROGRAMMED ANALYSIS
+*          OF AN ABNORMAL TERMINATION CONDITION.
+CXTBEGIN       DS   OF          (FULLWORD ALIGNMENT)
+CXTPCW         DS   CL8         CANCELLED PROGRAM'S PCW
+CXTPROGRAM     DS   CL8         NAME OF CANCELLED PROGRAM
+*                                  (X'00' IF UNABLE TO OBTAIN
+*                                  BUFFER DURING CANCEL
+*                                  PROCESSING!)
+CXTFLAGS        DS   XL1         VALUE IN PFBCXTOPTS AT TIME
+*                                  OF PROGRAM CANCELLATION
+CXTSPARE        DS   CL47        (RESERVED)
+CXTREGS         DS   CL64        REGISTER'S OF PROGRAM
+*                                  AT TIME OF PROGRAM CANCEL
+CXTMSGLIST      DS   0X          CANCEL MSGLIST
+CXTMSGID        DS   CL4         MESSAGE IDENTIFIER
+CXTMSGISSUER    DS   CL6         MESSAGE ISSUER
+CXTMSGLENGTH    DS   H           MESSAGE LENGTH
+CXTMSG          EQU  *           MESSAGE BEGINS HERE
+               CSECT
```

Delete Record from Indexed File

Syntax

```
[label]  DELETE  UFB={(register)} [,COND={integer                }]
                     {expression} [      {absolute expression}]
```

Restrictions

The file specified must be open for IO or SHARED mode processing. In IO mode, the last function on this file must have been a successful READ with the HOLD option. In SHARED mode, the record to be rewritten must be held as a result of a READ with the HOLD option.

Function

To delete the last record read from an indexed file on disk. Normally, control is returned to the instruction location following the DELETE macroinstruction. If the record to be deleted is not held, if the file is not an indexed file, or if the DELETE function is not allowed for the current 'open mode', control is returned to the I/O error return address as specified in the UFB, with the normal return address in register 0. If the I/O error return address in the UFB contains all binary zeroes when an error occurs, the program is abnormally terminated.

File status bytes in the UFB are set as follows for DELETE:

- Success — UFBFS1=0, UFBFS2=0
- I/O error — UFBFS1=3, UFBFS2=0
- Invalid function or function sequence — UFBFS1=9, UFBFS2=5

Operand Descriptions

UFB= The address of a User File Block. It may be presented as a register specification, where the register is assumed to contain the UFB address, or as an expression not in parentheses, in which case the word addressed is assumed to begin the UFB.

COND= If specified, the number or absolute expression becomes the first operand of the JSCI instruction by which the DELETE function is entered. Thus the DELETE is made conditional. COND=15 is the default. Register 1 is loaded with the UFB address in any case.

Examples

```
 LAB1      DELETE     UFB=(R2)
+LAB1      LR         1,R2
+          JSCI       15,12(1)

 LAB2      DELETE     UFB=DSKUFB,COND=7
+LAB2      LA         1,DSKUFB
+          JSCI       7,12(1)
```

Destroy Intertask Message Port

Syntax

```
[label] DESTROY PORT= {(register) }
                      { expression}
                      { 'literal' }
```

Function

Deactivates the intertask message receipt port with the specified port name, which must have been activated by the same task by means of the CREATE macroinstruction.

Return codes are placed in the word on the stack top as follows:

0 - Successful.
4 - One or more messages were not received, and are lost; otherwise successful.
8 - No such message buffer was allocated by this task.

Operand Description

PORT= The four-character name of a message receipt port. May be specified as a register in parentheses pointing to the port name, as a literal in single quotes which is the port name, or as an expression addressing a 4-byte field containing the port name.

Example

```
 LAB1  DESTROY  PORT=(R1)
+LAB1  PUSHC    0(4,0),0(R1)
+      SVC      38 (DESTROY)
```

Dismount Disk or Tape Volume

Syntax

```
[label] DISMOUNT VOLUME=  {address   }      , TYPE = {DISK}
                          {(register)}               {TAPE}
                          {'string'  }

                ,NODISPLAY=   {YES}
                              {NO }
```

Restrictions

None.

Function

To request the dismounting of a disk or tape volume. If the volume referenced is a tape volume, then it is also rewound and unloaded.

DISMOUNT issues a return code to the user program in the stack top word which indicates the success/failure/status of the operation (see DISMOUNT SVC description for possible values).

Operand Descriptions

VOLUME= The name of the volume which is to be dismounted. It may be specified as as a register in parentheses pointing to the volume name, as a character string in single quotes which is the volume name, or as an expression addressing a 6-byte field containing the volume name. This operand is required.

TYPE= Indicates whether the volume is a disk or a tape volume. Valid values are DISK and TAPE. This operand is optional and defaults to DISK.

NODISPLAY= If YES is supplied, indicates that no messages are to be displayed on the user's workstation; the operator console messages must be used to coordinate physical dismounting. The default is NO.

Examples

```
   LAB    DISMOUNT VOLUME='VOL444',TYPE = DISK
+  LAB    PUSHN    0,8              GET TWO WORDS ON THE STACK
+         MVC      2(6,15),*+10     SET VOLUME NAME
+         B        *+10             BRANCH AROUND CONSTANT
+         DC       CL6'VOL444'      VOLUME NAME
+         MVI      0(15),X'00'      SET FLAG FOR DISK VOLUME
+         MVI      1(15),X'00'      SET BYTE 1 to ZEROES (RESERVED)
+         SVC      41 (DISMOUNT)    ISSUE SVC


   LAB    DISMOUNT VOLUME=(R4)
+  LAB    PUSHN    0,8              GET TWO WORDS ON THE STACK
+         MVC      2(6,15),0(R4)    SET VOLUME NAME
+         MVI      0(15),X'00'      SET FLAG FOR DISK VOLUME
+         MVI      1(15),X'00'      SET BYTE 1 TO ZEROES (RESERVED)
         SVC      41 (DISMOUNT)    ISSUE SVC


   LAB    DISMOUNT       VOLUME=TAPEVOL,TYPE=TAPE
+  LAB    PUSHN    0,8              GET TWO WORD ON THE STACK
+         MVC      2(6,15),TAPEVOL  SET VOLUME NAME
+         MVI      0(15),X'80'      SET FLAG FOR TAPE VOLUME
+         MVI      1(15),X'00'      SET BYTE 1 TO ZEROES (RESERVED)
+         SVC      41 (DISMOUNT)    ISSUE SVC
```

Extract Data From System Control Blocks

Syntax

```
[label] EXTRACT FORM=   {LIST }
                        {BRIEF}
                        {FULL }
                        {PCPCW}

AREA=a1,          NRES=a2,          DYVAL=a3,         SYSVOL=a4,
SYSLIB=a5,        SYSWORK=a6,       VERSION=a7,       OCNT=a8,
WS=a9,            STACK=a10,        EXFLGS=a11,       RDFLGS=a12,
WTFLGS=a13,       SEG2BUF=a14,      USERID=a15,       USERNAME=a16,
EXTPRIOR=a17,     PCPCW=a18,        TASK#=a19,        TASKTYPE=a20,
CURVOL=a21,       CURLIB=a22,       WORKLIB=a23,      SPOOLIB=a24,
SEG2SIZE=a25,     STATIC=a26,       PRINTER=a27,      RUNVOL=a28,
RUNLIB=a29,       INVOL=a30,        INLIB=a31,        OUTVOL=a32,
OUTLIB=a33,       PRNTMODE=a34,     FILECLAS=a35,     LINES=a36,
PROGVOL=a37,      PROGLIB=a38,      WORKVOL=a39,      SPOOLVOL=a40,
PRTCLASS=a41,     FORM#=a42,        WSIO=a43,         TAPEIO=a44,
DISKIO=a45,       PRINTIO=a46,      OTIO=a47,         PICOUNT=a48,
POCOUNT=a49,      SICOUNT=a50,      SOCOUNT=a51,      ETIME=a52,
PTIME=a53,        DEVICE=(a54,a55), VOLUME=(a56,a57),
OTASK=(a58,a59),  TAPEVOL=(a60,a61), DEVCNT=a62,
ATOETRT=a63,      ETOATRT=a64,      DEVLIST=(a65,a66,a67),
VERSION=a68,      SYSPAGE=69,       CPU=a69,          HZ=a70,
UEXFLGS=a71,      URDFLGS=a72,      UWTFLGS=a73,      CLUSTER=a74,
JOBQUEUE=a75,     JOBCLASS=a76,     JOBLIMIT=a77,     JOBNAME=a78,
DLPNAME=(a79,a80),                  DLPDEV#=(a81,a82),
CDISKET=a83,      VOLVCB=(a84,a85)
```

Restrictions

None.

Function

Retrieves data from system control blocks that may be useful to user programs.

The following outputs for FORM = BRIEF, FORM = FULL, or FORM = PCPCW are placed in the area addressed by the AREA= operand, by ascending addresses:

FORM=BRIEF:

(1) Total physical area in bytes not currently resident (4 bytes)

(2) Number of files which a task may have open simultaneously (2 bytes)

(3) Workstation number associated with requesting task, or -1 if none (2 bytes)
(4) Remaining stack space in bytes after return from EXTRACT (4 bytes)

FORM = FULL:

(1) Total physical area in bytes not currently resident (4 bytes)
(2) Number of files which a task may have open simultaneously (2 bytes)
(3) Workstation number associated with requesting task, or -1 if none (2 bytes)
(4) Remaining stack space in bytes after return from EXTRACT (4 bytes)
(5) One day in clock units (4 bytes)
(6) System default library's volume name (6 bytes)
(7) System default library name (8 bytes)
(8) Task's default printer number, or -1 if none (2 bytes)
(9) User program library volume (6 bytes)
(10) User program library name (8 bytes)
(11) Current file-access bit map for 'execute' access (from Program File Block (PFB)) (4 bytes)
(12) Default non-output volume for 'OPEN' (6 bytes)
(13) Default non-output library name (8 bytes)
(14) Current file-access bit map for 'read' access (from Program File Block (PFB)) (4 bytes)
(15) Default output volume for 'OPEN' (6 bytes)
(16) Default output library name (8 bytes)
(17) Current file-access bit map for 'update' access (from Program File Block (PFB)) (4 bytes)
(18) Number of segment 2 buffer pages currently available (2 bytes)
(19) Print output mode (Spooled (S), Keep (K), Hold (H), or On-line (O)) (1 byte)
(20) Default output file-protection class, or blank (1 byte)
(21) User logon identification (3 bytes)
(22) Task current paging priority (from Task Control Block) (1 byte)
(23) Suggested lines-per-page for print files (1 byte)
(24) Operating System version number (Packed number 'VVRRPP' where VV is the version, RR is the revision, and PP is the patch level) (3 bytes)

FORM = PCPCW

(1) Program Control Word (PCW) at time of most recent program exception for which a user exit was specified (8 bytes)

Operand Descriptions

FORM= BRIEF is used to request four items as described above. Thus the output area must be at least 12 bytes long. FULL is used to request all 20 items listed above, and thus the output area must be at least 98 bytes long. PCPCW is used to request the value of the Program Control Word (PCW) current when a program exception occurred for which an exit routine was provided, and is intended for use in such a routine. (Its use at other times results in undefined and irrelevant output.) The output area must be at least 8 bytes long. LIST is used when a list of needed items is supplied.

AREA= Specifies the address of the output area, either as an expression addressing that area, or as a register expression in parentheses, where the register contains the address of the area. Not valid with FORM=LIST.

The following operands are used with FORM = LIST only. The operand specifies the address of an area to receive the corresponding data item.

SYSTEM-WIDE INFORMATION:

NRES=	Total physical area in bytes not currently resident (4 bytes)
DYVAL=	One day in clock units (4 bytes)
SYSVOL=	System default library's volume name (6 bytes)
SYSLIB=	System default library name (8 bytes)
SYSWORK=	System work library (paging files, system task queues, etc.) which BACKUP skips (8 bytes)
VERSION=	Operating System version number (Packed VVRRPP, where VV is the version, RR is the revision, and PP is the patch level) (3 bytes)
SYSPAGE=	System paging library name (8 bytes)
DEVCNT=	Device number (=255) of highest-numbered I/O device on the system (4 bytes)
ATOETRT=	ASCII-to-EBCDIC translate table (256 bytes). See TR instruction in Principles of Operation Manual for use.
ETOATRT=	EBCIDIC-to-ASCII translate table (256 bytes). See TR instruction in Principles of Operation Manual for use.
CDISKET=	Device number of system's central diskette (2 bytes)
CPU=	Current CPU ID (2 bytes)
HZ=	A/C line frequency (2 bytes)

TASK-RELATED INFORMATION:

OCNT=	Number of files which current task may have open simultaneously, excluding files already open (2 bytes)
WS=	Workstation number associated with requesting task, or -1 if none (2 bytes)
STACK=	Remaining stack space in bytes after return from EXTRACT (4 bytes)
EXFLGS=	Current file-access bit map for 'execute' access (from Program File Block (PFB)) (4 bytes)
RDFLGS=	Current file-access bit map for 'read' access (from Program File Block (PFB)) (4 bytes)
WTFLGS=	Current file-access bit map for 'update' access (from Program File Block (PFB)) (4 bytes)
UEXFLGS=	User's base file-access bit map for 'execute' access (from user's Extended Task Control Block (ETCB)) (4 bytes)
URDFLGS=	User's base file-access bit map for 'read' access (from user's Extended Task Control Block (ETCB)) (4 bytes)
UWTFLGS=	User's base file-access bit map for 'update'access (from user's Extended Task Control Block (ETCB)) (4 bytes)
SEG2BUF=	Number of segment 2 buffer pages currently available (2 bytes)
USERID=	User logon identification (3 bytes)
USERNAME=	User name (from system user list) (24 bytes)
EXTPRIOR=	Task's current paging priority (from Task Control Block) (1 byte)
PCPCW=	Program Check Old PCW for last program check (8 bytes)
TASK#=	Unique task identifier (4 bytes)
TASKTYPE=	Task type ('F' for foreground, 'FS' for dedicated foreground system task, 'B' for background task, and 'BS' for dedicated background system task) (2 bytes)
CURVOL=	Volume where current program resides (6 bytes)
CURLIB=	Library in which current program resides (8 bytes)
WORKLIB=	Work library name constructed from user ID or BG task number (8 bytes)
SPOOLIB=	Spool library name constructed from user ID or BG task number (8 bytes)
JOBNAME=	Name of background job (8 bytes)
SEG2SIZE=	Length of segment 2, in bytes (4 bytes)
STATIC=	Pointer to beginning of static areas for current program (May be useful in re-establishing addressability in a CEXIT routine) (4 bytes)

USER DEFAULTS (May be set by using the SET SVC or the SET Command Processor function):

PRINTER=	Task's default printer number, or -1 if none (2 bytes)
RUNVOL=	User program library volume (used by Command Processor RUN function) (6 bytes)
RUNLIB=	User program library name (used by Command Processor RUN function) (8 bytes)
INVOL=	Default non-output volume for OPEN (6 bytes)
INLIB=	Default non-output library (8 bytes)
OUTVOL=	Default output volume for OPEN (6 bytes)
OUTLIB=	Default output library (8 bytes)
PRNTMODE=	Default print output mode (1 byte)
FILECLAS=	Default output file-access protection class, or blank (1 byte)
LINES=	Suggested lines-per-page for print files (1 byte)
PROGVOL=	User program volume name used by LINK SVC (6 bytes)
PROGLIB=	User program library used by LINK SVC (8 bytes)
WORKVOL=	Default work volume (6 bytes)
SPOOLVOL=	Default spool volume (6 bytes)
PRTCLASS=	Default print class for print files (A-Z) (1 byte)
FORM#=	Default form number for print files (0-254) (1 byte)
JOBQUEUE=	Default job status (Run (R) or Hold (H)) (1 byte)
JOBCLASS=	Default job class (A-Z) (1 byte)
JOBLIMIT=	Default job CPU time limit (4 bytes)

RUN STATISTICS:

WSIO=	Count of workstation I/Os this run (4 bytes)
DISKIO=	Count of disk I/Os this run (4 bytes)
TAPEIO=	Count of tape I/Os this run (4 bytes)
PRINTIO=	Count of printer I/Os this run (4 bytes)
OTIO=	Count of I/Os for other devices not included under WSIO, DISKIO, PRINTIO, or TAPEIO (4 bytes)
PICOUNT=	Program pagein count (4 bytes)
POCOUNT=	Program pageout count (4 bytes)
SICOUNT=	System pagein count (4 bytes)
SOCOUNT=	System pageout count (4 bytes)
ETIME=	Elapsed time of run since command processor initiation, in hundredths of seconds (4 bytes)
PTIME=	Processor time of run since command processor initiation, in hundredths of seconds (4 bytes)

The following operands are used with FORM=LIST only. Two addresses are supplied. The first address specifies further input, and the second address specifies as area to receive the corresponding data.

DEVICE= Input: Device address (1 byte)
Output:
(1) Device class (1 byte)
(2) Device type (1 byte)
(3) Usage - 'EX' (exclusive, 'SH' (shared)' or 'DT' (detached) (2 bytes)
(4) Task identifier of device owner, or -1 if none (4 bytes)
(5) Volume name number of removable volume (disk or tape only). Blank if nothing mounted. (6 bytes)
(6) Volume name of fixed volume (disk only). Blank if nothing mounted. (6 bytes)
(7) 4 bytes of binary zeroes (reserved)

VOLUME= Input: Volume Name (6 bytes)
Output:
(1) Device address, or -1 if volume not mounted (1 byte)
(2) Volume type: 'F' for fixed, 'R' for removable, or blank if not mounted (1 byte)
(3) Label type: 'SL' (standard label), 'NL' (no label), or blank if not mounted (2 bytes)
(4) Usage - 'SH' (shared), 'RR' (restricted removal), 'PR' (protected), 'EX' (exclusive), or blank
(5) Task identifier of volume mounter, or -1 if none (4 bytes)
(6) Blocks per cylinder (2 bytes)
(7) Maximum transfer in bytes (2 bytes)
(8) Cylinders per volume (2 bytes)
(9) Cylinders per physical volume, including bad or unused blocks (2 bytes)
(10) Number of files open on this volume (2 bytes)
(11) Sector type (diskette only): soft sector (S), hard sector (H)
(12) Addressing in effect (diskette only): Non-standard (N), Standard (S)
(13) Unused (2 bytes)

OTASK= Input: Task identifier (4 bytes)
Output:
(1) Workstation device number of task specified, or -1 if none (1 byte)

(2) Current user ID for task specified, or blank if none (3 bytes)
(3) Current user name for task specified, or blank if none (24 bytes)
(4) Type ('F', 'FS', 'B', 'BS') of task specifed (see TASKTYPE) (2 bytes)
(5) 18 bytes of binary zeroes (reserved)

TAPEVOL= Input: Volume name (6 bytes)
Output:
(1) Device address, or -1 if volume not mounted (1 byte)
(2) 1 byte of binary zeroes (reserved)
(3) Density, BPI in binary: 556, 800, or 1600 (2 bytes)
(4) Label type: 'AL' (ANSI), 'NL' (no label), 'IL' (IBM label), or blank if volume not mounted (2 bytes)
(5) Usage: 'SH' (shared), 'EX' (exclusive), or blank if not mounted (2 bytes)
(6) Task identifier of tape mounter, or -1 if none (4 byts)
(7) Current file sequence number (2 bytes)
(8) 6 bytes of binary zeroes (reserved)

DLPNAME= Input: Name of Data Link Processor (as specified in the SYSGEN procedure)

NOTE:
The output area will be all zeroes if the specified DLP name is invalid.

Output:
(1) Bit map of devices on DLP (4 bytes)
(2) First device on DLP (2 bytes)
(3) Type of DLP (1 = 22V06-1, 2 = 22V06-2, 3 = 22V06-3) (1 byte)
(4) Number of lines (RS-232) controllable by the DLP (1 byte)
(5) Microcode file status (X'00' if stopped, X'80' if loaded) (1 byte)
(6) Reserved for future use (3 bytes)
(7) Microcode file name (8 bytes, zero if not loaded)
(8) Microcode library name (8 bytes, zero if not laoded)
(9) Microcode volume name (6 bytes, zero if not loaded)
(10) Reservation status of DLP (X'80' if reserved, X'00' if not reserved)
(11) Task number of the task which reserved the DLP (3 bytes)

DLPDEV# Input: Device address (2 bytes)

NOTE:
For the DLPDEV# operand, the output area will contain zeroes if the specified device address is invalid.

Output:
(1) Device status flag (X'80' if open, X'40' if reserved, zero otherwise)
(2) Task number of the task which reserved the DLP, or zero if device is unreserved (3 bytes)
(3) Name of the DLP on which the device is SYSGENed (4 bytes)

CLUSTER= Input: Device number (2 bytes)

NOTE:
This operand is used for obtaining the device number of the archiver diskette on the same cluster as the device number which is specified as input. (If more than one archiver diskette is on the cluster, then the device number that is returned belongs to the archiver whose device number is next in sequence.)

Output:
(1) Device number of the archiver diskette, or zero if none (2 bytes)
(2) Unused (14 bytes)

VOLVCB= Input: Volume name (6 bytes)

Output: Volume Control Block (VCB) address (4 bytes)

The DEVLIST operand has three suboperands. The first address specifies further input, the second address specifies an area to receive the corresponding data, and the third suboperand is the length of the output area (specified as an expresssion or register in parentheses). Note that the maximum number of device addresses in the device list will be two less than the output length specified.

DEVLIST= Input: Device class, as in EXTRDDEVCLASS (1 byte)
Output:
(1) Total number of devices for specified device class (1 byte)
(2) Number of device addresses supplied (1 byte)
(3) Device address list (1 byte for each device address)

Examples

```
    LAB1    EXTRACT  FORM=BRIEF,AREA=(R3)
+   LAB1    PUSH     0,R3          AREA
+            MVI      0(SP),0       FORM=BRIEF
+            SVC      28 (EXTRACT)

    LAB2    EXTRACT  INLIB=A1,INVOL=(R1)
+   LAB2    PUSH     0,R1          AREA FOR INVOL
+            PUSHA    0,11          IDENTIFIER
+            PUSHA    0,A1          AREA FOR INLIB
+            PUSHA    0,12          IDENTIFIER
+            PUSHA    0,2           COUNT OF ITEMS
+            MVI      0(15),3       FORM=LIST
+            SVC      28 (EXTRACT)

    LAB3    EXTRACT  OUTLIB=A1,VOLUME=(A2,(R1))
+   LAB3    PUSHA    0,A1          AREA FOR OUTLIB
+            PUSHA    0,15          IDENTIFIER
+            PUSHA    0,1           COUNT OF ITEMS
+            MVI      0(15),3       FORM=LIST
+            SVC      28 (EXTRACT)
+            PUSHA    0,A2          VOLSER ADDRESS
+            PUSH     0,R1          AREA FOR OUTPUT
+            PUSH     0(15),24      CURRENT OUTPUT LENGTH
+            PUSHA    0,51          IDENTIFIER
+            PUSHA    0,1           COUNT OF ITEMS
+            MVI      0(15),4       FORM=LIST WITH ADDITIONAL INPUT
+            SVC      28 (EXTRACT)

    LAB4    EXTRACT  DEVLIST=(A2,(R1),12)
+   LAB4    PUSHA    0,A2          DEVICE CLASS ADDRESS
+            PUSH     0,R1          AREA FOR OUTPUT
+            MVI      0(15),12      SPECIFIED OUTPUT LENGTRH
+            PUSHA    0,59          IDENTIFIER
+            PUSHA    0,1           COUNT OF ITEMS
+            MVI      0,(15),4      FORM=LIST WITH ADDITIONAL INPUT
+            SVC      28 (EXTRACT)
```

Generate Selected Parameter Group Control List Fields

Syntax

```
[label]  FMTLIST  [LABELPFX='prefix',]

         {'Keyword',({'displayed-value'}                          }
         {          { absolute-length }                           }
         {                                                        }
         {          [,CHAR ]                                      }
         {          [,INT  ]                                      }
         {          [,NUM  ]                                      }
         {          [,AN   ]                                      }
         {          [,HEX  ]                                      }
         {          [,UCHAR]                                      }
         {          [,ANL  ]                                      }
         {                                                        }
         {          [,line-advance][,space-advance]),             }
         {                                                        }
         { {TEXT,    }                                            }
         { {textname,} ('displayed-text'                          }
         {                  [,line-advance][,space-advance]       }
         {                  [,'CENTER'][,'RIGHT'])                }

         ['Keyword2',(...),...]

         [,PREVIEW = {YES}]
                     {NO}
```

Function

Generates Field Format Control Blocks for use in a Parameter Group Control List as addressed by SVC GETPARM and SVC PUTPARM. The function is identical to that of the KEYLIST macroinstruction, except that the first eight bytes of the Parameter Group Control List are not generated. Thus, a PRNAME may not be specified.

Operand Descriptions

As for the KEYLIST macroinstruction, except that PRNAME may not be specified.

Example

```
 LAB1        FMTLIST     LABELPFX='XXX',                    X
           TEXT1,('HEADING'),                               X
           TEXT,('SUBHEADING'),                             X
           'LIST',('NO',AN)
+LAB1          EQU         *
+              DC          HL1'0'   PF KEY
+              DC          HL1'3'   FIELD COUNT
+              DC          HL1'1,0,-1.6'
+TEXT1         DC          C'HEADING'
+              DC          HL1'1,0,-1,9'
+              DC          C'SUBHEADING'
+XXXLIST       DC          HL1'1,0,0,1'
+              DC          CL8'LIST'
+              DC          C'NO'
```

Free Buffer Space

Syntax

```
[label]  FREEBUF  BUFLOC={(register) }[,LENGTH=(register)]
                         { expression}
```

Restrictions

For use by certain supervisor call routines and Data Management System routines only.

Function

To deallocate a buffer area allocated by GETBUF. The buffer area at the address (in segment 2, as provided by a preceding GETBUF) specified by the BUFLOC operand and for the length specified by the LENGTH operand is made available for reallocation by GETBUF. The contents of this area should be considered unreliable after the FREEBUF has been issued.

A return code is left on the stack:

0 - Buffer deallocated
4 - Invalid buffer address
8 - Invalid buffer length

Operand Descriptions

BUFLOC= The address of a buffer allocated by GETBUF. This must be presented as a register specification in parentheses, where the register is assumed to contain the buffer address, or as a buffer address expression not in parentheses.

LENGTH= A register specification in parentheses where the register contains the buffer length. The length must be a multiple of 2048, and must be the same as that requested by GETBUF. LENGTH of 2048 is assumed if no LENGTH= operand is supplied.

Example

```
 LAB1     FREEBUF    BUFLOC=(R1)
+LAB1     PUSHA      0,2048
+         PUSH       0,R1
+         SVC        6 (FREEBUF)
```

Deallocate Heap Storage

Syntax

```
[label]    FREEHEAP   SIZE=      (register)
                     ,LINKLEV=   address
                      ,BUFLOC=   {(register)}
                                 {address   }
                    ,POOLNAME=   {address   }
                                 {'string'  }
                          [,ROOTLEV] [,SEARCH] [,DELETE]
```

Restrictions

A stack with the stack top addressed by the general register 15 must be available.

Function

Deallocates heap storage previously allocated by the GETHEAP SVC.

Operand Descriptions

SIZE= The size of the block to be allocated. Specified as a register in parentheses where the register contains the size of the block in the low-order three bytes. When the deletion of an entire subpool is specified (i.e., the DELETE parameter is specified), the SIZE parameter is ignored.

BUFLOC= Start address of the buffer/block to be deleted. Specified as a register in parentheses containing the start address of the buffer/block in the low-order three bytes, or as an address expression pointing to a 4-byte field which contains the start address of the buffer/block in the low-order three bytes. When the deletion of an entire subpool is specified (i.e., the DELETE parameter is specified), the BUFLOC parameter is ignored. BUFLOC must be specified if DELETE is not specified.

LINKLEV= Link level at which to start searching for the specified subpool. A value of '0' indicates the current link level, a value of '1' is the parent, and so on. Specified as an address expression pointing to a one-byte field containing the link level in binary. Default is 0 (i.e., current link level).

POOLNAME= Name of the subpool to be searched/deleted. Specified as a 1- to 8-byte character string in quotes which is the name of the subpool, or as an address expression pointing to an 8-byte character string not in quotes. Blank names are not permitted. Trailing blanks are insignificant. There is a system-defined default poolname for each link level.

ROOTLEV If specified, sets the LINKLEV parameter to 255 (X'FF'), which indicates the lowermost link level. Any other value specified with LINKLEV= is ignored if ROOTLEV is specified.

SEARCH If specified, a backward search for the subpool is to be initiated starting from the LINKLEV specified. Default is no backward search.

DELETE If specified, asks for the deletion of an entire subpool. The SEARCH parameter is ignored if DELETE is specified. The BUFLOC parameter and SIZE parameter are ignored if DELETE is specified.

Example

```
 LAB1    FREEHEAP SIZE=(3),POOLNAME=NAMELOC,BUFLOC=START,ROOTLEV
+LAB1    PUSHN    0,16     RESERVE STACK SPACE FOR PARAMETERS
+        XC       (16,15),0(15)      INITIALIZE PARAMETER SPACE
+        MVC      8(8,15),NAMELOC    MOVE POOLNAME TO STACK
+        STCM     2,B'0111',1(15)    MOVE SIZE PARAMETER TO STACK
+        MVC      5(3,15),START      MOVE START ADDRESS TO STACK
+        OI       4(15),X'FF'        SET LOWERMOST LINK LEVEL
+        SVC      57 (FREEHEAP)
```

Get Buffer Space

Syntax

[label] GETBUF [LENGTH=(register)]

Restrictions

For use by certain supervisor call routines and Data Management System routines only.

Function

To allocate a data management buffer area on a 2048-byte (page) boundary. Buffer space is allocated from the low-address end of segment 2. Control register 2 (the stack limit word) may be modified by this function and by FREEBUF. Two words are stacked as output of this function:

- If a buffer is allocated, the top word of the stack contains binary zero, and the next word contains the buffer address.
- If a buffer cannot be allocated, the top word of the stack contains four in binary, and the next word's contents are undefined.
- If the requested length is not a multiple of 2K, the top word of the stack contains eight in binary, and the next word's contents are undefined.

Operand Description

LENGTH= A register specification in parentheses where the register contains the buffer length. Only lengths which are multiples of 2048 are valid. If the operand is omitted, LENGTH of 2048 is assumed.

Example

```
 LAB1     GETBUF
+LAB1     PUSHN     0,4
+         PUSHA     0,2048
+         SVC       5 (GETBUF)
```

Allocate Heap Storage

Syntax

```
[label]     GETHEAP     SIZE=     (register)

                      ,LINKLEV=   address

                      ,POOLNAME=  {address }
                                  {'string'}

                          [,ROOTLEV] [,ALIGN] [,SEARCH] [,CREATE]
```

Restrictions

A stack with the stack top addressed by general register 15 must be available.

Function

This macro provides a user-level memory management feature known as heap storage allocation. Heap storage is storage independent of the system stack that can be allocated dynamically. The GETHEAP facility is a generalization of the GETBUF macro and SVC (for allocating page-aligned buffers) with the following additional features:

1. Any size block can be allocated. It is not necessary for the size to be a multiple of 2K. Any size is automatically rounded up to the nearest 8-byte multiple.

2. Blocks may be put into different 'subpools'. Advantages of subpooling are that clustering of areas allocated from the same subpool will tend to occur, and that blocks in a given subpool may be allocated in separate calls of the GETHEAP macro and then deallocated together by one FREEHEAP call.

3. All subpools associated with a specified link level are released automatically on UNLINK for that level.

Because blocks are automatically released at program termination, present GETBUF users are encouraged to convert to GETHEAP.

Operand Descriptions

SIZE= The size of the block to be allocated. Specified as a register in parentheses where the register contains the size of the block in the low-order three bytes.

LINKLEV= Link level at which to start searching for the specified subpool. A value of '0' indicates the current link level, a value of '1' is the parent, and so on. Specified as an address expression pointing to a one-byte field containing the link level in binary. Default is 0 (i.e., current link level).

POOLNAME= Name of the subpool to be searched/created. Specified as a one- to eight-byte character string in quotes, or as an address of an eight-byte field containing an eight-byte character string not in quotes. Blank names are not permitted. Trailing blanks are insignificant. There is a system-defined default poolname for each link level.

ROOTLEV If specified, sets the LINKLEV parameter to 255 (X'FF'), which indicates the lowermost link level. Any other value specified with LINKLEV= is ignored if ROOTLEV is specified.

ALIGN When specified, requests 2K-alignment for all blocks which are a multiple of 2K in size. This parameter is ignored for blocks which are not a multiple of 2K. The default is no alignment.

SEARCH If specified, a backward search for the subpool is initiated starting from the LINKLEV specified. The default is to no backward search.

CREATE If specified, asks for the creation of a new subpool with the name given by the POOLNAME= parameter and at the link level given by LINKLEV. The SEARCH parameter is ignored if CREATE is specified.

Example

```
 LAB1      GETHEAP   SIZE=(2),POOLNAME='POOL',LINKLEV=LEVL,CREATE
+LAB1      PUSHN     0,16    RESERVE STACK SPACE FOR PARAMETERS
+          XC        0(16,15),0(15)     INITIALIZE PARAMETER SPACE
+          MVC       8(8,15),*+10       MOVE POOLNAME TO STACK
+          B         *+12
+          DC        CL8'POOL'
+          STCM      2,B'0111',1(15)    MOVE SIZE PARAMETER TO STACK
+          MVC       4(1,15),LEVL       MOVE  LINK LEVEL PARAMETER TO
+*                                       STACK
+          OI        0(15),X'40'        SET THE CREATE FLAG
+          SVC       56 (GETHEAP)
```

Get Parameters

Syntax

```
[label]  GETPARM  [I ,]FORM={REQUEST}
                  [ID,]     {SELECT }
                  [R ,]     {ACK    }
                  [RD,]     {SYSHDR }
                            {OPR    }

                   ,KEYLIST={(register1)}     ,MSG={(register2)}
                            {expression1}          {expression2}

                   ,DEVICE={(register3)} [,PFKEYS={(register4)}          ]
                            {expression3} [       {expression4}          ]
                                          [       {(ENTER, expression4)}]
```

Function

To solicit formatted information from a procedure body or from the user's workstation. Fields for which values are requested are identified by a two-level name (PRNAME and Keyword) specified in the Parameter Group Control List addressed by the KEYLIST operand. The procedure body in effect, if any, is the preferred source of values for a type 'I' (initial) request. In the absence of a matching name in a procedure, the user is solicited at the workstation. A type 'ID' (initial defaulted) request solicits from the procedure body only. A type 'R' (respecification) request solicits from the workstation only. A type 'RD' request normally solicits no information from the workstation or from a procedure body, but updates the procedure's temporarily stored information for use by reference from a later procedure step.

The MSGLIST macroinstruction may be used to generate a message for display. The KEYLIST macroinstruction may be used to generate the Parameter Group Control List addressed by the 'KEYLIST=' operand of GETPARM.

The total number of lines utilized by KEYLIST and MSGLIST displays may not exceed 18. None of these lines may be longer than 79 characters, excluding end-of-line characters.

The user should consult the GETPARM Supervisor Call description and the KEYLIST macroinstruction description in this document for details concerning the function of this macroinstruction.

Operand Descriptions

I
ID
R
RD

Indicates the type of request. If not specified, 'I' is the default value.

FORM=

Valid options are shown in the syntax specification; these are:

REQUEST - Request for information (default option);
SELECT - Request for selection;
ACK - Request for acknowledgment;
SYSHDR - Request for information with no PRNAME displayed;
OPR - Request for operator action.

For details, consult the GETPARM SVC description.

MSG=

The address of a message in the format specified in the GETPARM Supervisor Call description. This is the form of message generated by the MSGLIST macroinstruction. It may be presented as a register specification in parentheses, where the register contains the message address, or as an expression not in parentheses, where the expression addresses the message. A message is always required, but the message text may be of length zero.

KEYLIST=

The address of a keyword specification and display formatting list (format control list), in the format specified in the GETPARM Supervisor Call description. This is the format produced by the KEYLIST macroinstruction. This operand may be presented in the same ways as the 'MSG=' operand. A format control list is always required, but may if desired have no Field Format Control Blocks (Keyword or text specifications).

DEVICE=

Device number in binary in the low byte of the specified register or at the byte in memory specified by the expression. Required if FORM=OPR is specified; displayed when FORM=OPR only.

PFKEYS= If supplied as a single suboperand not in parentheses or if supplied in parentheses with the word ENTER (as shown in the syntax above), the designated expression is to be used in a 4-byte A-type address constant indicating which program function keys are to be accepted. The high-order bit corresponds to program function key 1, the low-order bit to program function key 32. Bits on indicate keys to be accepted. This expression may be preceded by 'ENTER,' in which case the ENTER key is also accepted. Otherwise, if the 'PFKEYS=' operand is supplied, the ENTER key is not accepted.

May also be specified by designating a register in parentheses where the register contains the program function key map.

If the 'PFKEYS=' operand is not supplied, the following keys are accepted:

FORM=REQUEST	ENTER only
FORM=SELECT	All PF keys and ENTER
FORM=ACK	ENTER only
FORM=SYSHDR	ENTER only
FORM=OPR	ENTER and PF key 16 only

Example

```
 LAB1     GETPARM     KEYLIST=(R2),MSG=LAB2
+LAB1     PUSH        0,R2
+         LA          0,LAB2
+         PUSH        0,0
+         SVC         20 (GETPARM)

 LAB2     MSGLIST     '1234','TXTEDT','OPTIONS AS FOLLOWS:'

 LAB3     KEYLIST     PRNAME='OPT',                          X
         'LIST',('NO',AN,1,0),                               X
         'DISPLAY',('YES',AN,1,0),                           X
         'LINECNT',('50',INT,1,0)
```

Halt I/O Operation

Syntax

1. [Label] HALTIO PRINTER = { (register) }
   = { integer }
   = { expression}

2. [Label] HALTIO OFB = { (register) }
   = { expression}

Restrictions

Intended for use by system routines and those user programs which must control I/O operations thru "XIO" (Execute Physical I/O). NOT to be used by programs using normal DMS for I/O.

HALTIO must not be issued unless an unCHECKed XIO is currently outstanding. The user program must always wait for the HALTIO to complete by issuing a subsequent CHECK I/O macroinstruction.

Functions

1. To terminate multi-line (especially block-oriented) print I/O requests to a printer.
2. To terminate an outstanding I/O request to/from a file which is not necessarily a printer output file (especially telecommunications files).
3. To terminate an outstanding volume-oriented I/O request to/from a disk.

HALTIO issues a Return Code in the stack top word for the "PRINTER" form of the macroinstruction. This Return Code corresponds to the condition code set by the HIO machine instruction (see "VS Principles of Operation").

HALTIO does not issue a Return Code for the "OFB" or "VCB" forms of the macroinstruction. The stack is cleared by the SVC.

Operand Descriptions

PRINTER - The device number of the printer whose current I/O is to be terminated. This number must be in the range 0 - 255 and may be specified as a register in parentheses containing the device number in binary in its low-order position as an integer which is the device number in decimal, or as an expression addressing a one-byte field containing the device number in binary.

OFB - The address of the Open File Block for the outstanding I/O. This form is used for file-oriented (regular) I/O and may reference any file/device pairing. This operand may be specified as a register in parentheses or as a four-byte data item defined in the user program.

NOTE: The two operands are, of course, mutually exclusive.

Examples

```
 LAB    HALTIO PRINTER=(R3)
+LAB    PUSHA 0,0                    GET ONE WORD OF ZEROS ON THE
             STACK
+       STC  R3,3(,15)               PUT PRINTER NUMBER IN LOW-
        ORDER BYTE
+       SVC  12  (HALTIO)            ISSUE SVC

 LAB    HALTIO PRINTER=3
+LAB    PUSHA 0,3                    PUSH PRINTER NUMBER ONTO STACK
+       SVC  12  (HALTIO)            ISSUE SVC

 LAB    HALTIO PRINTER=PBLKID
+LAB    PUSHA 0,0                    GET ONE WORD OF ZEROS FROM THE
             STACK
+       MVC  3(1,15),PBLKID          PUT PRINTER NUMBER IN
                                          LOW-ORDER BYTE
+       SVC  12   (HALTIO)           ISSUE SVC

 LAB    HALTIO OFB=(R4)
+LAB    PUSH 0,R4                    PUSH OFB ADDRESS ONTO STACK
+       MVI  0(15),X'80'             FLAG AS OFB/VCB TYPE PARMLIST
+       SVC  12  (HALTIO)            ISSUE SVC
```

Generate Parameter Group Control List

Syntax

```
[label]  KEYLIST  PRNAME='name',[LABELPFX='prefix',]

         {'Keyword1',({'displayed-value'},                    }
         {             { absolute-length }                    }
         {                                                    }
         {             [,CHAR ]                               }
         {             [,INT  ]                               }
         {             [,NUM  ]                               }
         {             [,AN   ]                               }
         {             [,HEX  ]                               }
         {             [,UCHAR]                               }
         {             [,ANL  ]                               }
         {                                                    }
         {           [,line-advance][,space-advance]),        }
         {                                                    }
         {                                                    }
         {                                                    }
         { {TEXT,   }                                         }
         { {textname} ('displayed-text'                       }
         {               [,line-advance][,space-advance]      }
         {               [,'CENTER'][,'RIGHT'])               }

         ['Keyword2',(...),...]

         [,PREVIEW = {YES}]
                     {NO}
```

Restrictions

Intended for use in conjunction with the GETPARM macroinstruction. See that macroinstruction and the GETPARM Supervisor Call description.

Function

Generates a data structure suitable for use as a parameter group control list with SVC GETPARM (object of 'KEYLIST=' operand of the GETPARM macroinstruction).

Operand Descriptions

PRNAME=	A name identifying the parameter group. May be up to eight characters in length; characters may be alphabetic and/or numeric (first character must be alphabetic).
LABELPFX=	A character string in quotes which will be prefixed to each 'Keyword' name and the resulting string used to label each corresponding keyword block. The label is placed on the line-advance byte. Thus the flag byte is at the location specified by this label +2, and the receiving field ('displayed-value') is at this location +12. This operand is optional.
'Keyword'	A name of up to eight alphabetic and/or numeric characters enclosed in single quotes, identifying a specific parameter within the group. Specification of 'Keyword' is mutually exclusive with specification of TEXT or text name.
'displayed-value'	A character string in single quotes containing the default value for this specific parameter. Single quotes to appear in the string must be represented by two consecutive single quotes. The receiving field length is then the length of this string. Specification of 'displayed-value' is mutually exclusive with specification of absolute-length.
absolute-length	An absolute expression must be provided defining the length of the receiving field for this parameter. Specification of absolute-length is mutually exclusive with specification of 'displayed-value.'
CHAR	Any character accepted in receiving field.
INT	Only unsigned integers accepted.
NUM	Numbers (with optional decimal point and/or leading sign) accepted.

AN	Letters (including national characters #, @, and $) and numerals accepted. GETPARM will convert letters to upper case.
HEX	Only numerals and letters A-F accepted. Letters A-F converted to upper case. (Leading and trailing blanks are accepted in any format except alphanumeric [AN], wherein only trailing blanks are accepted.)
UCHAR	Any characters accepted. Lowercase letters converted to uppercase.
ANL	Letters (including national characters #, @, and &) and numerals accepted. GETPARM will convert letters to uppercase. The first character must not be a number.
line-advance	A positive number, zero, or omitted (in which case 1 is assumed). If nonzero, the keyword or text is displayed starting in column 2 (plus the value in the space-advance suboperand) of the line which is the specified number of lines in advance of the current line. If zero, line advancing does not occur, and one space (plus the number of spaces specified by the space-advance suboperand) appears on the workstation screen between the previous displayed values and this keyword or text.
TEXT Textname	Indicates that embedded text, rather than a keyword and receiving field, is supplied in the next operand. If a nonquoted textname is provided, it may be used to symbolically address the beginning of the actual text field in the Parameter Group Control List, i.e., the label "textname" is generated for the specified text field.

space-advance

If specified as either an expression with a value no less than zero or greater than 78, or omitted (in which case, zero is assumed), the value of space-advance plus 1 is the number of spaces that will appear on the workstation screen between either the previous field (if zero line-advance) or the left side (if nonzero line-advance) and the keyword or text of the current field.

The space-advance may also be specified in three alternative formats:

'Ann'

"nn" represents one or two digits with a value no less than "2" and no greater than "80" that indicate the "absolute" column in which the field is to begin. The appropriate field-advance value is calculated and placed in the control block.

'CENTER'

The appropriate field-advance value is calculated (and placed in the control block) such that the field is centered within the 80 column workstation screen line.

'RIGHT'

The appropriate field-advance value is calculated (and placed in the control block) such that the field is right-justified on the 80 column workstation screen line.

Regardless of how the space-advance is specified, a MNOTE is generated if an attempt is made to generate a workstation line over 80 characters in length or if an absolute, centering, or right-adjust request cannot be honored.

'displayed-text'

A character string in quotes to be displayed as embedded text.

'Keyword2',...

Any number of keyword or embedded text operands may be supplied.

PREVIEW

If YES is specified, the screen display specified by the macroinstruction operands will be printed in the source listing. NO is the default.

Example

```
 LAB1      KEYLIST     PRNAME='OPT',LABELPFX='A',                X
          'LIST',('NOb',AN),                                     X
          'DISPLAY',('YES',AN,0,5),                              X
          TEXT,('NUMBER OF LINES'),                              X
                      'LINECNT',('50',INT,0,5)
+LAB1      DC         CL8'OPT'            PRNAME
+          DC         HL1'0'              PF KEY
+          DC         HL1'4'              FIELD COUNT
+ALIST     DC         HL1'1,0,0,2'        LA,SA,FLAGS,LGTH-1
+          DC         CL8'LIST'           KEYWORD
+          DC         C'NOb'              FIELD
+ADISPLAY  DC         HL1'0,5,0,2'        LA,SA,FLAGS,LGTH-1
+          DC         CL8'DISPLAY'        KEYWORD
+          DC         C'YES'              FIELD
+          DC         HL1'1,0,-1,14'      LA,SA,FLAGS,LGTH-1
+          DC         C'NUMBER OF LINES'
+ALINECNT  DC         HL1'0,5,1,1'        LA,SA,FLAGS,LGTH-1
+          DC         CL8'LINECNT'
+          DC         C'50'
```

Link to Another Program or Subprogram

Syntax

```
[label]  LINK  [EP='literal'          ]
               [EPLOC=Address of name]

                  [,SYSTEM][,NOFAIL][,LOADONLY]

                  [,LIBRARY={Address  }]
                            {'literal'}

                  [,VOLUME ={Address  }]
                            {'literal'}
```

Restrictions

A stack, with stack top addressed by general register 15, must be available to the issuer.

Function

Pushes the name parameter and flag byte onto the top of the stack and invokes the specified program by SVC LINK. If the LIBRARY and VOLUME operands are specified, they override the user program library and volume specified on the SET or RUN command. The invoked program may return to the invoker by means of the RETURN macroinstruction. Execution of the LINK macroinstruction pushes status information onto the stack, as well as the 'static' areas of the LINKed-to program as described in section 2.2.1.1.3 of this document. Addresses on the stack to be passed to an invoked program should be placed in a parameter list addressed by register R1. This parameter list may not be in the reentrant program segment (segment 1). On entry to a LINKed-to program, register R14 addresses the (new) 'static' area base (if any) as defined in Part I of this document. Register R1 addresses the user's argument list (that is, R1 is preserved across the LINK). Any user program exception exit previously set by SVC PCEXIT is eliminated, but is restored when an UNLINK is issued to return to the LINKed-from program.

If the specified file exists but is not a program file, the file, library, and volume names are pushed onto the stack and LINK initiates execution of the system's Procedure Interpreter, which then attempts to interpret the file as a procedure.

Operand Descriptions

EP= A name of up to eight characters, enclosed in quotes, which is used in conjunction with the current program library name (as a member name in that library) to form a complete file name, which is then sought and the corresponding file invoked as a program if found. If not found, the supplied name is used in conjunction with the system library name, and the resulting file name is sought.

EPLOC= A byte address, which must not be in a user's program segment (segment 1), at which there is a character string of length eight giving the name of the member to be concatenated with the current program library name or system library name (as for the EP= operand). This must be specified in a form allowable in the D2(B2) fields of the SS-type assembler instruction format.

LIBRARY= A byte address at which there is a character string of length eight giving the overriding user program library name for use on this LINK and LINKs nested below this link, or a character literal in single quotes giving this name. The previous default library name becomes effective again upon UNLINK to this LINK issuer.

VOLUME= Name of volume containing the overriding user program library, specified as for the LIBRARY operand.

SYSTEM Specifies that the user's program library is not to be searched for the requested member. Only the system library is searched.

NOFAIL Specifies that the program is not to be terminated by the CANCEL SVC in the event that the requested program is not found, or cannot be acquired or executed, but rather that control is to be returned to the address of the LINK SVC instruction plus six bytes (next sequential instruction address plus four). This option is intended primarily for Command Processor use. A code is returned in the top word of the stack to indicate the specific error condition (see LINK SVC description).

LOADONLY — Specifies that after the new program or subprogram has been made addressable in segment 1, and all initialization of segment 2 areas (including the Link Return List) has been accomplished, control will be returned to the address of the LINK SVC plus 10 bytes, instead of being passed to the new program. The new program's entry point address will be in register zero when control is returned to the LINK issuer. The LINK SVC must be issued from segment 0 if this option is to be used.

Examples

```
 LAB1      LINK    EP='PROG1'
+LAB1      PUSHC   0(16),LG001
+          B       LG002
+LG001     DC      X'00',CL8'PROG1',XL7'0'
+LG002     SVC     4 (LINK)

 LAB2      LINK    EPLOC=PNAME,SYSTEM
+LAB2      PUSHN   0,16
+          MVI     0(SP),B'10000000'     FLAG BYTE
+          MVC     1(8,SP),PNAME         PROGRAM NAME
+          SVC     4 (LINK)
```

Supply Program Parameters

Syntax

```
(1)  [label]  LINKPARM  PUT,                    {DISPLAY}
                                                {ENTER  }

                                  ,PRNAME=      'string'

                                ,REFERLABEL=    {address   }
                                                {'string'  }
                                                {(register)}

                                   ,FMTLIST=    {address   }
                                                {(register)}

                                   [   ,AID=    {address   }]
                                   [            {'string'  }]
                                   [            {(register)}]

                                   [ ,PFKEY=    {ENTER}     ]
                                   [            {1-32 }     ]

                                   [ ,LABEL=    {address   }]
                                   [            {'string'  }]
                                   [            {(register)}]

                                   [,REPEAT=    {address   }]
                                   [            {'string'  }]
                                   [            {(register)}]
                                   [            {NO        }]
                                   [            {YES       }]

(2)  [label]  LINKPARM CLEANUP  [,REFERLABEL=   {address   }]
                                [               {'string'  }]
                                [               {(register)}]

(3)  [label]  LINKPARM  REFER

                ,MERGE [, REMOVE ]  ,FMTLIST=   {address   }
                                                {(register)}

                ,NOMERGE            ,REFERLABEL= {address   }
                                                 {'string'  }
                                                 {(register)}
```

Restrictions

None.

Function

The LINKPARM macro accesses the functions of the PUTPARM SVC (SVC 33). The primary function (the PUT function) is to supply parameters to another program's GETPARMs before issuing the LINK SVC to invoke that program. The second function (the CLEANUP function) is to deallocate the various internal data structures created by the PUT function. The third function (the REFER function) is to allow the calling program access to any parameters which the user may have changed at GETPARM time (the MERGE option), or to return the address of a previously created and labelled FMTLIST (the NOMERGE option). See the PUTPARM SVC description for further detail on each of these functions of the PUTPARM SVC.

Note that both the PUTPARM macro and the LINKPARM macro call the PUTPARM SVC (SVC 33). The PUTPARM macro allows only the parameterization of another program (the PUT function), while the LINKPARM macro accesses all the functions of the PUTPARM SVC. Users of the PUTPARM macro are encouraged to use the LINKPARM macro because of the more extensive functionality. The PUTPARM macro is kept for compatibility with existing programs.

Operand Descriptions

PUT — Enables a program to supply parameters to a GETPARM issued by another program. The parameters to be supplied to the GETPARM are contained in a format list (FMTLIST), created with the FMTLIST macroinstruction. The program issuing the LINKPARM PUT must link via the LINK SVC to the program issuing the GETPARM. A program may not use LINKPARM PUT to pass parameters to its own GETPARM.

DISPLAY
ENTER — If DISPLAY is specified, requests a workstation transaction when the FMTLIST supplied to the linked-to program is accessed. If ENTER is specified, suppresses a workstation transaction when this FMTLIST is accessed. The default is ENTER.

PRNAME= — A name of up to 8 alphanumeric characters which identifies the PRNAME to be associated with the FMTLIST being supplied to the linked-to program or the new PRNAME to be used if this is a backward reference. Specified as a character string in quotes.

REFERLABEL= A name of up to 8 alphanumeric characters which identifies a previously labeled FMTLIST. This parameter is used to "backward reference" a previously created FMTLIST. The backward reference facility allows a program to reuse the (possibly updated) parameters of a labelled FMTLIST. (See PUTPARM SVC description for further detail regarding backward reference facility). Specified as an expression addressing an 8-byte field containing the name of the FMTLIST, as a register in parentheses pointing to an 8-byte field containing the name of the FMTLIST, or as a character string in quotes which is the name of the FMTLIST. Note that, for the PUT function, REFERLABEL= and FMTLIST= are mutually exclusive . For the CLEANUP function, REFERLABEL= specifies a particular FMTLIST to be deallocated. For the MERGE option, REFERLABEL= contains the name of the source FMTLIST, while FMTLIST= is the address of the destination FMTLIST.

FMTLIST= The address of the FMTLIST to be used. The FMTLIST is created by the FMTLIST macro. (See the FMTLIST macro description for further detail). Optionally the address of a KEYLIST+8 may be supplied. Specified as an expression addressing a FMTLIST, or as a register in parentheses containing the address of the FMTLIST. Note that, for the PUT function, REFERLABEL= and FMTLIST= are mutually exclusive.

AID= The AID (Attention ID) character of a PFkey to be passed to the GETPARM. AID characters are 'A'-'P' (i.e., PFkeys 1-16, respectively), 'a'-'p' (i.e., PFkeys 17-32, respectively), and '@' (i.e., the ENTER key). Specified as an expression addressing a one-byte field containing the AID character, as a register in parentheses pointing to a one-byte field containing the AID character, or as a character string in single quotes which is the AID character. Note that AID= and PFKEY= are mutually exclusive.

PFKEY= A PFkey to be passed to the GETPARM. PFKEY= may be a number from 1 through 32, or the word ENTER. PFKEY= must be a character string not in quotes. Note that PFKEY= and AID= are mutually exclusive.

LABEL= A FMTLIST may be labelled for later use by the backward reference and override facilities. (See PUTPARM SVC description for further detail.) A name of up to 8 alphanumeric characters is used to label the saved FMTLIST. May be specified as an expression addressing an 8-byte field containing the label, or as a register pointing to an 8-byte field containing the label.

REPEAT= Normally, no two GETPARM requests access the same FMTLIST. A FMTLIST may be declared to be for repeated use via the macro parameter REPEAT=. If REPEAT=NO (or is missing), the FMTLIST will be used only once. If REPEAT=YES, the FMTLIST will be used until it is removed. If REPEAT=n, the FMTLIST will be used n+1 times (initial use + n repeats). May also be specified as an expression addressing a 2-byte binary repeat count or as a register in parentheses pointing to a 2-byte binary repeat count. The value of the repeat count can range from 1-32768.

CLEANUP If CLEANUP is specified, the various internal structures created by the PUT function are deallocated. If no REFERLABEL is provided, all FMTLISTs created at this level and above are removed. If a REFERLABEL is provided, only the labeled FMTLIST will be removed. If the CLEANUP option is used, REFERLABEL is the only other parameter which may be suppplied.

REFER Allows previously created and used FMTLISTs at the current link level to be accessed.

MERGE The MERGE option of the REFER function allows the "merging" of an updated 'used' labelled FMTLIST with a program-designated FMTLIST in the user's address space. The contents of the FMTLIST addressed by REFERLABEL= (the source) are merged into the FMTLIST addressed by FMTLIST= (the destination). Fields which are present in the source, but not in the destination, are ignored. Fields present in the destination but not in the source are left unchanged.

NOMERGE Requests LINKPARM to return the address (in the Segment 2 buffer) of the FMTLIST referenced by the REFERLABEL= operand (i.e., a previously created and labelled FMTLIST). The address is returned on the stack.

REMOVE Requests LINKPARM to remove (CLEANUP) the source FMTLIST after performing the merge. This option is only available with MERGE.

Examples

```
 LAB1   LINKPARM PUT,DISPLAY,PRNAME='OLDPRNAM',FMTLIST=FMTL1,
                       LABEL='FOO1',AID='A'
+LAB1   DS      0H                          PLACE HOLDER FOR LABEL
+       PUSHC   0(8),=CL8'FOO1'             FMTLIST LABEL
+       PUSHC   0(8),=CL8'OLDPRNAM'         PRNAME
+       PUSHA   0,0                         UNUSED
+       PUSHA   FMTL1                       FMTLIST
+       MVI     FMTL1,C'A'                  AID CHARACTER
+       PUSHA   0,0                         INITIAL FLAG BITS
+       OI      0(15),X'80'                 DISPLAY FLAG
+       SVC     33   (PUTPARM)


 LAB2   LINKPARM PUT,PRNAME='NEWPRNAM',REFERLABEL='FOO1',PFKEY=1
+LAB2   DS      0H                          PLACE HOLDER FOR LABEL
+       PUSHC   0(8),=CL8''                 NULL LABEL FOR FMTLIST
+       PUSHC   0(8),=CL8'NEWPRNAM'         PRNAME
+       PUSHC   0(8),=CL8'FOO1'             REFERLABEL
+       PUSHA   0,0                         INITIAL FLAG BITS
+       MVI     1(15),65                    AID CHARACTER
+       SVC     33   (PUTPARM)


 LAB3   LINKPARM REFER,NOMERGE,REFERLABEL='FOO1'
+LAB3   PUSHC   0(16),=CL16''               NULL LABEL AND PRNAME
+       PUSHC   0(8),=CL8'FOO1'             REFERLABEL
+       PUSHA   0,0                         INITIAL FLAG BITS
+       OI      0(15),X'40'                 REFER FLAG
+       SVC     33   (PUTPARM)
```

Log Off Interactive Terminal

Syntax

```
[label]  LOGOFF
```

Function

The LOGOFF macroinstruction generates the code to issue the appropriate SVC call and parameter list to effect a "logoff by program request."

Operand Descriptions

No operands are required.

Example

```
 LAB2          LOGOFF
+LAB2          PUSHA 0,0          NULL
+              PUSHA 0,0          PARAMETERS
+              SVC   43           (LOGOFF)
```

Mount Disk or Tape Volume

Syntax

```
(1) [label] MOUNT    DISK= {(register)}     ,VOLUME= {(register)}
                           {integer   }              {'string'  }
                           {address   }              {address   }

                  ,LABEL= {SL }        ,BLP= {NO }        ,USAGE= {SH }
                          {NL }              {YES}                {RR }
                                                                  {PR }
                                                                  {EX }

                ,VOLTYPE= {R}        ,SPOOL= {NO }        ,WORK= {NO }
                          {F}                {YES}              {YES}

                    ,NSA= {NO }  ,NODISPLAY= {NO }   ,NOMESSAGE= {NO }
                          {YES}              {YES}              {YES}

(2) [label] MOUNT    TAPE= {(register)}     ,VOLUME= {(register)}
                           {integer   }              {'string'  }
                           {address   }              {address   }

                  ,LABEL= {AL }        ,BLP= {NO }        USAGE= {SH }
                          {NL }              {YES}                {EX }
                          {IL }

                ,NOMESSAGE= {NO }
                          {YES}
```

Restrictions

None.

Functions

1) To request the mounting of a disk volume on the indicated device with the specified label, usage, type, SPOOL file, and Work file attributes.

2) To request the mounting of a tape volume on the indicated device with the indicated device with the specified label attributes.

MOUNT issues a return code to the user program in the stack top word which indicates the success/failure/status of the operation (see MOUNT SVC description).

Operand Descriptions

DISK= A number between 0 and 255 which is the system-defined device number of the disk unit on which the volume is to be mounted.

TAPE= A number between 0 and 255 which is the system defined device number for the tape unit on which the volume is to be mounted.

DISK and TAPE may be specified as a register in parentheses containing the device number in binary in its low-order position, as an integer not in quotes which is the device number in decimal, or as an expression addressing a one-byte field containing the device number in binary. One of these operands is required and they are mutually exclusive.

VOLUME= The name of the volume which is to be mounted. It may be specified as a register in parentheses pointing to the volume name, as a character string in single quotes which is the volume name, or as an expression addressing a 6-byte field containing the volume name. This operand is required.

THE FOLLOWING OPERANDS ARE OPTIONAL

BLP= This operand instructs the system to bypass label processing/checking and should be specified with care. Valid values are YES and NO. The default is to NO.

LABEL= Denotes the type of volume label present on a volume. Valid values are:

SL - Standard WANG VS labels.
NL - No labels are present on the volume.
AL - Standard ANSI-type labels.
IL - Standard IBM-type labels.

The default for a disk volume is to SL; for a tape volume the default is to AL. Note that SL is valid for disk volumes only and that AL and IL are valid for tape volumes only.

USAGE= Denotes volume access and dismounting restrictions. Note that dismounting restrictions also apply to remounting with different attributes. Valid values are:

SH - Shared: Volume may be accessed and dismounted by any user.

RR - Restricted Removal: Volume may be accessed by any user but dismounted by the mounting user only.

PR - Protected: Files on the volume may be read by any user but updated and dismounted by the mounting user only.

EX - Exclusive: Volume may be accessed and dismounted by the mounting user only.

Default is to SH for both disk and tape volumes. RR and PR are valid for disk volumes only.

VOLTYPE= Denotes the type of disk volume being mounted as either fixed or removable. Valid values are F and R respectively, with the default being to R. This operand is valid for disk volumes only.

SPOOL= Denotes whether the volume is to be included in the list of volumes scanned when the system creates a SPOOL (Print) file for a user whose values are YES and NO, with the default being to NO. This operand is valid for disk volumes only.

WORK= Denotes whether the volume is to be included in the list of volumes scanned when the system creates a Work file for a user whose default work volume has not been SET. Valid values are YES and NO, with the default being to NO. This operand is valid for disk volumes only.

NSA= If YES is specified, indicates that the volume to be mounted follows non-standard addressing conventions (soft-sectored diskette only). The default is to NO.

NODISPLAY= If YES is specified, indicates that no messages are to be displayed on the user's workstation; the operator console messages must be used to coordinate physical mounting. The default is to NO.

NOMESSAGE= If YES is specified, indicates that the volume to be mounted is already on the disk or tape drive. No MOUNT message will be displayed, and the Volume Control Block (VCB) information is updated from the volume label. The default is to NO.

Examples

```
LAB     MOUNT   DISK=(R1),VOLUME='SYSTEM',LABEL=SL,USAGE=SH,          X
                VOLTYPE=F,SPOOL=NO,WORK=YES
+LAB    PUSHN   0,8                 GET TWO WORDS ON THE STACK
+       STC     R1,1(,15)           SET DEVICE NUMBER
+       MVC     2(6,15),*+10        SET VOLUME NAME
+       B       *+10                BRANCH AROUND CONSTANT
+       DC      CL6'SYSTEM'         VOLUME NAME
+       MVI     0(15),B'00010001'   SET FLAGS
+       SVC     30 (MOUNT)          ISSUE SVC


LAB     MOUNT   DISK=DISKVOL,VOLUME=(R4)
+LAB    PUSHN   0,8                 GET TWO WORKDS ON THE STACK
+       MVC     1(1,15),DISKVOL     SET DEVICE NUMBER
+       MVC     2(6,15),0(R4)       SET VOLUME NAME
+       MVI     0(15),B'00000000'   SET FLAGS
+       SVC     30 (MOUNT)          ISSUE SVC

LAB     MOUNT TAPE=28,VOLUME=TAPEVOL,LABEL=IL,USAGE=EX
+LAB    PUSHN   0,8                 GET TWO WORDS ON THE STACK
+       MVC     0(1,15),*+10        SET DEVICE NUMBER
+       B       *+6                 BRANCH AROUND CONSTANT
+       DC      AL1(28)             DEVICE NUMBER
+       MVC     2(6,15),TAPEVOL     SET VOLUME NAME
+       MVI     0(15),B'01010000'   SET FLAGS
+       SVC     30 (MOUNT)          ISSUE SVC
```

Generate Display Message

Syntax

```
[label]  MSGLIST  msg#,issuer,'message-segment-1'
                              [,'message-segment-2',...]
```

Restrictions

Intended for use in conjunction with the GETPARM and CANCEL macroinstructions. See those macroinstructions and the corresponding Supervisor Call descriptions.

Function

Generates a data structure suitable for use as the object of 'MSG=' operands of the GETPARM and CANCEL macroinstructions.

Operand Descriptions

msg#	Up to four characters enclosed in single quotes, normally a message number, to be displayed with the message. The msg # is displayed on row 1 of the workstation screen.
issuer	Up to six characters enclosed in single quotes, normally an identification of the issuing routine, to be displayed with the message. The issuer is displayed on row 1 of the workstation screen.
'message-segment-n'	Message text in single quotes to be displayed on a single line. May be repeated as often as required, to define additional lines to be displayed. No line may be over 79 characters long. The message may contain single quotes (apostrophes). Message text is displayed beginning on row 3 of the workstation screen.

Example

```
 LAB1      MSGLIST    '123','ISSUER','LINE 1','LINE 2'
+LAB1      DC         CL4,'123'
+          DC         CL6'ISSUER'
+          DC         AL2(6+6+1)
+          DC         C'LINE 1'
+          DC         X'0D'  NEW LINE
+          DC         C'LINE 2'
```

Open a File

Syntax

```
[label]  OPEN  UFB= {(register)}
                   {expression}

               [,MODE= {OUTPUT}]
                       {INPUT }
                       {IO    }
                       {EXTEND}
                       {SHARED}

               [ ,{NOGETPARM} ]
                  {NODISPLAY}

               [,EXIT= {(register)          }]
                       {absolute expression}

               [,PLOG= {YES }]
                       { NO }
```

Restrictions

None.

Function

Prepares a file for processing by Data Management System functions. The User File Block (UFB) is normally created prior to OPEN by means of the UFBGEN macroinstruction. The OPEN macroinstruction includes provision for optional modification of the 'Open mode' flags of the UFB. If the file was already open to the issuing task, no additional OPEN processing occurs. In this case the file remains open in the mode specified in the Open File Block addressed by this UFB.

Operand Descriptions

UFB= The address of a User File Block, which must be specified either as a register designation in parentheses, where the register is assumed to contain the UFB address, or as a UFB address expression not in parentheses. If omitted, only the MVI for open mode modification and the SVC instruction will be generated.

MODE= Specifies a value to be placed in the UFB to designate an open mode. This is done before

the OPEN Supervisor Call is issued. This operand is optional.

NOGETPARM — Causes a GETPARM type "RD" to be issued rather than a type "I". This suppresses user interaction and causes procedure-supplied parameters to be ignored. This option should be used only when run-time parameters have already been obtained through a program-issued GETPARM. In this case, the programmer should also use the OPEN exits which enable the program to handle error conditions.

NODISPLAY — Causes a GETPARM type "ID" to be issued rather than a type "I". This suppresses user interaction as long as the values supplied in the UFB or through a procedure are not lexically in error.

NOTE:

User interaction will occur even with the NOGETPARM or NODISPLAY options if a field is semantically in error (e.g., an invalid device type).

EXIT= — A value indicating which file assignment problems should cause control to be returned to the issuing program rather than cause generation of a user interaction via a GETPARM (Type "R"). See description of OPEN SVC for possible values. May be specified as a register designation in parentheses, or as an absolute expression not in parentheses. The value in the low-order byte of the register, or the value of the expression, is stored in the high-order byte of the OPEN parameter word on the stack.

PLOG= — If YES is specified, a file prologue will be created when the file is OPENed. Valid only for Word Processing files. This operand must be specified when the file is OPENed in OUTPUT mode (MODE=OUTPUT) in order for the file prologue to be identified with the file to be created. The default is to NO.

Example

```
 LAB1     OPEN     UFB=(R2),MODE=INPUT
+LAB1     MVI      44(R2),X'20'  INPUT MODE
+         PUSH     0,R2
+         SVC      0 (OPEN)
```

Modify Program Exception Exit Status

Syntax

```
1.   [label]  PCEXIT  SET, [(list),ADDRESS={(register)}]
                                          {expression}]

2.   [label]  PCEXIT  {RESET }
                      {CANCEL}
```

where 'list' may contain any of the following, separated by commas:

OP	Operation
PO	Privileged Operation
EX	Execute
PR	Protection
AD	Addressing
SP	Specification
DA	Data
FIO	Fixed Point Overflow
FID	Fixed Point Divide
DO	Decimal Overflow
DD	Decimal Divide
SR	Supervisor Call Range
SO	Stack Overflow
FPO	Floating Point Overflow
FPU	Floating Point Underflow
SI	Significance
FPD	Floating Point Divide

To specify that all types of program exception interrupts are to be intercepted, specify 'ALL' instead of a 'list'.

Function

SET — To specify user-program exception handling for the listed program interruptions. The previous program exception handling status, if any, is saved for use by the RESET function.

RESET — Restores user-program exception handling status to its state before the most recent SET function, if there was a 'PCEXIT SET' issued in the current program. The most recent status is discarded.

CANCEL — Removes all user-program exception handling in the current program (since a LINK from another program or Command Processor program initiation). All such status is discarded.

Operand Descriptions

ADDRESS= This and the '(list)' operand may be specified only with the 'SET' operand. A register specification in parentheses signifies that the register contains the exit address. An expression not in parentheses is evaluated to the exit address directly.

See above for other operand descriptions and 'list' suboperand descriptions.

Example

```
 LAB1      PCEXIT    SET,(FPU,SI),ADDRESS=(R1)
+LAB1      PUSHC     0(4,0),*+10
+          B         *+8
+          DC        BL4 '0000000000000000110' LIST
+          PUSH      0,R1          EXIT ADDRESS
+          MVI       0(SP),0       SET
+          SVC       31 (PCEXIT)
```

Protect a Disk File

Syntax

```
[label]   PROTECT   {   PLIST =  {expr }                          }
                    {              {(reg)}                         }
                    {                                              }
                    { { LIBRARY   }          ,LIBRARY=expr         }
                    { { FILE=expr }                                }
                    {                        ,VOLUME=expr          }
                    {                       [,OWNER=expr]          }
                    {                                              }
                    {                       [,FILECLAS=expr]       }

                                            [   , {RETPD=expr }]
                                            [     {EXPRDT=expr}]

                                            [,RESTRICT=  NO  ]
                                            [            YES ]
```

Function

To update the protection information (protection class, owner of record, and/or expiration date) for a disk file or a library of disk files on a volume. The structure of the Volume Table of Contents (VTOC) is not affected by the change. No file that is to have its protection information modified may be open when the PROTECT is attempted.

Return codes in binary in the top word of the stack indicate the result of the request:

```
Return Code =  0 - Protection status successfully changed
Return Code =  4 - Volume not mounted
Return Code =  8 - Volume used exclusively by other user
Return Code = 12 - All buffers in use, no protection change
Return Code = 16 - Library not found
Return Code = 20 - File not found
Return Code = 24 - Update access denied, no protection
                   change
Return Code = 28 - (unused)
Return Code = 32 - File in use, no protection change
Return Code = 36 - VTOC error!   FDX1 & FDX2 don't agree
Return Code = 40 - VTOC error!   FDX2 & FDR don't agree
Return Code = 44 - Invalid argument list address
Return Code = 48 - I/O error!    VTOC unreliable!
Return Code = 52 - Open or protected files bypassed in
                   protecting library
Return Code = 56 - Invalid new protection data
```

Note: If the PLIST= option is not utilized, it is the program's responsibility to pop 32 bytes off the stack beyond the return code word on the stack.

Operand Descriptions

PLIST= - The address of a PROTECT parameter list. (For a description of the format of the parameter list, please refer to the description of the PROTECT SVC in Chapter 6.) If PLIST= is specified, no other operand may be specified. If PLIST= is not specified, the macro generates code to dynamically build a parameter list on the stack prior to issuance of the PROTECT SVC.

LIBRARY - Indicates that the protection attributes of all files within the library specified are to be modified. Use of this operand is mutually exclusive with the FILE= operand.

FILE= - Specifies the address at which the file's name is located. May be specified as a character string delimited by single quotes, in which case, a constant is assumed. Use of this operand is mutually exclusive with the LIBRARY operand descrbed above.

LIBRARY= - Specifies the address at which the library's name is located. May be specified as a character string delimited by single quotes, in which case, a constant is assumed. This operand is required if PLIST= is not specified.

VOLUME= - Specifies the address at which the volume's name is located. May be specified as a character string delimited by single quotes, in which case, a constant is assumed. This operand is required if PLIST= is not specified.

OWNER= - If specified, indicates that the 3 byte "Owner of Record" protection attribute is to be modified and the address at which the new value is located. May be specified as a character string delimited by single quotes, in which case, a constant is assumed.

FILECLAS= - If specified, indicates that the 1 byte "File Class" protection attribute is to be modified and the address at which the new value is located. May be specified as a character string delimited by single quotes, in which case, a constant is assumed.

EXPRDT= - If specified, indicates that the "Expiration Date" protection attribute is to be modified and the address at which the new value (3 byte packed decimal, YYDDD+) is located. May be specified as a character string delimited by single quotes, in which case a constant is assumed. Use of this operand is mutually exclusive with use of the RETPD= operand.

RETPD= - If specified, indicates that the "Expiration Date" protection attribute is to be modified and the address at which a "Retention Period," in terms of days (3 byte packed decimal, 00DDD+) is located. May be specified as a character string delimited by single quotes, in which case a constant is assumed. Use of this operand is mutually exclusive with use of the EXPRDT= operand.

RESTRICT= - If NO is specified, or the operand is omitted, the PROTECT operation precedes utilizing current file access rights. If YES is specified, the operation is restricted, assuming only the file access rights of the user, ignoring any special access rights of the program.

Example

```
LAB1     PROTECT FILE=PROFILE,LIBRARY=PROLIBR,VOLUME=PROVOLUME, X
              OWNER='DOV',FILECLAS=PROCLAS,RETPD=PRORETPD
+LAB1    PUSHA 0,0
+        PUSHA 0,0
+        MVC   3(3,15),PRORETPD          RETENTION PERIOD
+        MVC   0(3,15),=CL3'DOV'         FILE OWNER OF RECORD
+        PUSHN 0,8
+        MVC   7(1,15),PROCLASS          FILE CLASS
+        MVI   6(15),B'00001111'
+        MVC   0(6,15),PROVOLUME         VOLUME
+        PUSHC 0(8),PROFILE              FILE
+        PUSHC 0(8),PROLIBR              LIBRARY
+        SVC   42 (PROTECT)
```

Supply Program Parameters

Syntax

```
[label]  PUTPARM     [ DISPLAY ]
                     [ ENTER   ]

                   , PRNAME = 'literal'

                   , FMTLIST = { (register) }
                               { expression }

                     [,LABEL = 'literal']
```

Restrictions

None.

Function

PUTPARM enables a program to supply parameters to a GETPARM issued by another program. The PUTPARM issuer must dynamically link to the program issuing the GETPARM via the LINK SVC. (A program may not use PUTPARM to parameterize its own GETPARMs.)

The parameters to be supplied to the GETPARM are contained in a format list (FMTLIST), created with a FMTLIST macroinstruction. When PUTPARM is issued, it verifies that the specified FMTLIST is in the proper format, then saves it in a Segment-2 buffer area for subsequent GETPARM use. PUTPARM also constructs a Parameter Reference Block (PRB) to save the label, PRNAME, display option, and certain other information.

When a GETPARM in the linked-to program is issued, it searches through the FMTLISTs in the Segment-2 buffer area. If a FMTLIST is found whose PRNAME matches the PRNAME of the GETPARM's KEYLIST, the FMTLIST parameter values are copied to the KEYLIST, thus supplying the required GETPARM parameters. A workstation transaction is suppressed if the 'ENTER' option is selected; otherwise, a GETPARM screen is displayed. PUTPARM returns to the issuer eight bytes of output on the top of the stack:

Bytes 0-3	return code
0	success
8	Bad FMTLIST supplied to this SVC
12	Error detected in previously constructed parameter reference blocks.
Bytes 4-7	Address of FMTLIST saved for GETPARM

Operand Descriptions

DISPLAY
ENTER — requests GETPARM to display or bypass displaying the screen.

PRNAME= — A name of up to 8 alphanumeric characters enclosed in quotes, identifying the PRNAME of the FMTLIST.

FMTLIST= — The address of the FMTLIST in the format specified in GETPARM SVC. (See the FMTLIST macroinstruction.)

LABEL= — A name of up to 8 alphanumeric characters enclosed in quotes. The label of this parameter reference block to be used by GETPARM.

Examples

```
LAB1  PUTPARM  ENTER,PRNAME='ABCDE'                              X
                ,FMTLIST=ADDR1,LABEL='XYZ'

LAB   PUTPARM  PRNAME='ABCDE',FMTLIST=(R2)
```

Read a Record

Syntax

```
                    [           ]*
[label]  READ      |HOLD  ,     | UFB={(register)}[,COND=number]
                    |REL        |     {expression}
                    |KEYED      |
                    |NODATA      |
                    |TABS        |
                    |MOD         |
                    |ALTERED     |
                    |CONNECTPARM|
                    |STATUS      |
                    [           ]
```

(* Various combinations in parentheses are allowed. See below.)

Function

To read a record from any file or device for which READ is supported by the Data Management System. This includes the special workstation READ functions (READ TABS, READ MOD). The function of the READ macroinstruction depends on the value of its first operand. Valid first operands for various device and file types are as follows:

- Fixed Length Consecutive disk files - omitted
  HOLD
  (HOLD,NODATA)
  REL
  (REL,HOLD)
  (REL,NODATA)
  (REL,HOLD,NODATA)

- Variable Length Consecutive disk files - omitted
  HOLD
  (HOLD,NODATA)

- Indexed disk files - omitted
  HOLD
  (HOLD,NODATA)
  KEYED
  (KEYED,HOLD)
  (KEYED,NODATA)
  (KEYED,HOLD,NODATA)

- Telecommunications - omitted
  CONNECTPARM
  STATUS

. Workstation files - {omitted } treated
{REL } identically

{MOD } treated
{(MOD,REL) } identically

{ALTERED } treated
{(ALTERED,REL)} identically

TABS

A file must have been opened in modes INPUT, IO or SHARED, or placed in temporary IO mode by the 'START IO' function, before attempting to READ the file. The record or workstation line, fields, or tab position indications are returned in the user's record area, as addressed by field UFBRECAREA of the UFB. For 'READ REL' or any workstation READ other than 'READ TABS', the records number within the file, or line number on the screen (from 1) to be read is taken from the word addressed by field UFBKEYAREA of the UFB. For 'READ KEYED', the key value is taken from the memory area beginning with the byte addressed by UFBKEYAREA, and extending for the number of bytes specified by UFBKEYSIZE. Descriptions of the functional effects of the various allowable suboperands (HOLD, REL, KEYED, NODATA, TABS, ALTERED, MOD) may be found in the Data Management System description (Part IV of this document) and in the operand descriptions below.

Invalid key and end-of-data conditions on a READ result in return to the address in UFBEODAD with the normal return point address in register 0 and file status bytes (UFBFS1, UFBFS2) set to the following ASCII characters:

10 - End of data
23 - Invalid key (no record found) on 'READ REL' or 'READ KEYED'

Other exceptional and error conditions result in return to the address in UFBERRAD with the normal return point address in register 0 and file status bytes (UFBFS1, UFBFS2) set to the following ASCII characters:

30 - Permanent I/O error
34 - Order check on workstation
95 - Invalid function sequence for block-level I/O
96 - Invalid data area location or alignment
97 - Invalid length for device
98 - Magnetic tape trailer label error (block count)

If UFBEODAD contains binary zero, the address in UFBERRAD is used for invalid key and end-of-data returns. If it too is zero, these conditions and I/O errors cause program terminations.

Operand Descriptions

REL — Indicates that the record or workstation line to be read is specified by the binary number (from 1) in the word addressed by UFBKEYAREA. Assumed for workstation files.

KEYED — Indicates that the record to be read from an indexed disk file is specified by the key value in bytes beginning at the address in UFBKEYAREA, and extending for the number of bytes specified in UFBKEYSIZE. The user's program should not modify UFBKEYSIZE.

HOLD — Indicates that the record read from a disk file may be rewritten by REWRITE. Must be specified in order to successfully complete a REWRITE of this record. For SHARED open mode, indicates that the record read from a disk file is not to be made available to any other simultaneously executing program which is sharing the file until either the record is rewritten, it is deleted, or another record in any shared file is read with HOLD requested. Note that this implies that a program may only HOLD one record at a time, no matter how many files are being shared.

NODATA — Indicates that the record requested is to be read from the file in the manner indicated by other suboperands (including the HOLD suboperand), but that the record is not to be placed in the user's record area as addressed by UFBRECAREA. The address of the record in the Data Management System buffer is placed in register 1. This option is not valid in SHARED open mode.

CONNECTPARM — Indicates that telecommunication line connection parameters are to be read.

STATUS — Indicates that telecommunication device status is to be read.

TABS — Indicates that current tab settings for the specified workstation are to be placed in the fifth through fourteenth bytes of the user's record area as addressed by UFBRECAREA. Values are column numbers 1-80 in binary. Zeroes indicate unset tab positions.

MOD — Indicates that the modifiable fields within the specified workstation line are to be placed in their corresponding positions in the user's record area as addressed by UFBRECAREA. Protected fields may or may not be read and placed in the user's record area, depending on the workstation model. If protected fields are not transferred, the corresponding positions in the user's record area are not changed.

ALTERED — Indicates that only those fields with selected-field tags set are to be placed in the user's record area, in positions corresponding to their screen positions. Other data on the user's record area remains unchanged. Field attribute characters of altered fields have their selected-field tags set on the corresponding field attribute characters in the user's record area.

UFB= — The address of a User File Block (UFB), which may be supplied as a register specification in parentheses, where the register contains the UFB address, or as an expression not in parentheses, where the word addressed is assumed to begin the UFB.

COND= — If specified, the number or absolute expression becomes the first operand of the JSCI instruction by which the READ function is entered. READ is thus made conditional. COND=15 is the default. Register 1 is loaded with the UFB address even when the condition is not satisfied.

Examples

```
 LAB1     READ    (REL,HOLD),UFB=(R3)
+LAB1     LR      1,R3                 SET REGISTER 1
+         MVI     0(1),B'00000101'     MODIFIERS
+         JSCI    15,0(,1)             READ FUNCTION

 LAB2     READ    UFB=UFBADDR
+LAB2     LA      1,UFBADDR            SET REGISTER 1
+         MVI     0(1),B'00000000'     MODIFIERS
+         JSCI    15,0(,1)             READ FUNCTION
```

Read File Descriptor Record(s)

Syntax

```
[label] READFDR PLIST={(register)}
                      { address  }

[label] READFDR LIBRARY={(register)}, FILE={(register)}, VOLUME={(register)},
                        {'string'  }       {'string'  }         {'string'  }
                        { address  }       { address  }         { address  }

                AREA={(register)}   [,FDR={1   }]      [,ALTLIB={(register)},
                     { address  }         {n   }                 {'string'  }
                                          {BOTH}                 { address  }

                ALTVOL={(register)}][,PLOG={NO  },      PAREA= {(register)}]
                       {'string'  }        {YES }              { address  }
                       { address  }        {ONLY }
```

Restrictions

The area addressed by PLIST must not be in the user's segment one (re-entrant code segment).

If any operand is specified by a string, the invoking user program must allow for the generation of a literal pool.

With the exception of specification by character strings, specifications of libraries and files must reference 8-byte fields, those for volumes must reference 6-byte fields.

Function

Allows user programs to locate a disk file on the specified volume and copy its File Descriptor Record(s) (label) into the memory location denoted by the AREA operand; also reads a WP file prologue and returns the file prologue into the specified AREA.

If PLIST is not specified, then operands FILE, LIBRARY, VOLUME, and AREA are required and the user program is responsible for popping 36 bytes (50 if ALTLIB is specified) beyond the FDR1 disk address word since the parameter list is dynamically built on the stack.

If an alternate search library (ALTLIB) is specified, then the values of the LIBRARY and VOLUME are modified as required to indicate in which library the file was found.

READFDR issues a return code to the user program in the stack top word which indicates the success/failure of the operation, and the disk address of the FDR1 in the next

stack word (this address is usable only if the read was successful, i.e., the return code equals zero). (See READFDR SVC for return codes and address .)

Operand Descriptions

PLIST= A user-generated parameter list as described in the READFDR SVC; specified as a register in parentheses pointing to the parameter list, or as an expression addressing the parameter list. If this operand is specified, then all other operands are ignored.

LIBRARY= The name of the primary library to be searched for the file in question; specified as a register in parentheses pointing to the library name, as a literal in single quotes which is the library name, or as an expression addressing a character string whose value is the library name.

FILE= The name of the file in question. It may be specified as for LIBRARY above.

VOLUME= The name of the volume on which the primary library resides. It may also be specified as for LIBRARY above.

AREA= A user receiving area for the obtained file descriptor record(s); 80 bytes if one FDR is requested and 160 if BOTH is specified. It may be specified as a register in parentheses pointing to the address of the receiving area, or as an expression addressing a 4-byte field containing the address of the receiving area.

FDR= This optional operand indicates which FDR(s) to read. If omitted, the default is to 1 (read FDR1 only).

1 -	Read the FDR1 only.
n -	Read the (n-1)th FDR2 only where "n" is integer 2 or higher. For example, "3" reads the second FDR2.
BOTH-	Read both FDR1 and the first FDR2.

ALTLIB= The name of a library to be searched if the file in question cannot be located in the library specified by the LIBRARY operand. It may be specified as for LIBRARY above. This operand is optional. However, if specified, then ALTVOL must be specified as well.

ALTVOL= The name of the volume on which the alternate search library resides. It may also be specified as for LIBRARY above. This operand is valid only in conjunction with ALTLIB.

PLOG= If YES, then read the WP file prologue along with any other options set. If ONLY, then the caller wants only the file prologue to be read. If NO is specified, then the caller does not request the file prologue be read.

If YES or ONLY is specified, then the caller must specify in the PAREA operand a receiving area for the file prologue.

PAREA= Indicates the address of the receiving area for the file prologue. May be specified as a register in parentheses containing the address of the receiving area, or as an address expression pointing to a four-byte area containing the address of the receiving area.

Examples

```
LAB    READFDR PLIST=(R4)
+LAB   PUSHA   0,0                GET ONE WORD OF ZEROES ON THE STACK
+      PUSH    0,R4               POINT TO PLIST WITH STACK TOP WORD
+      SVC     24   (READFDR)     ISSUE SVC

LAB    READFDR LIBRARY='SYSLIB',FILE=(R1),VOLUME=SYSVOL,AREA=FDRAREA,X
           FDR=BOTH,ALTLIB='SYSLIB2',ALTVOL=SYSVOL
+LAB   PUSHN   0,50               GET SPACE ON STACK FOR PLIST
+      MVC     0(8,15),=CL8'SYSLIB'   SET LIBRARY NAME
+      MVC     8(8,15),0(R1)      SET FILE NAME
+      MVC     16(6,15),SYSVOL    SET VOLUME NAME
+      MVI     22(15),X'04'       SET FLAG TO READ FDR1 AND 1ST FDR2
+      MVI     23(15),X'00'       (THIS FIELD NOT USED FOR FDR=BOTH)
+      MVC     24(4,15),FDRAREA   SET FDR RECEIVING AREA ADDRESS
+      XC      28(8,15),28(15)    (THIS FIELD RESERVED)
+      OI      22(15),X'08'       SET FLAG TO INDICATE ALTERNATES
+      MVC     36(8,15),=CL8'SYSLIB2'   SET ALTERNATE LIBRARY NAME
+      MVC     44(6,15),SYSVOL    SET ALTERNATE VOLUME NAME
+      PUSHA   0,0                GET ONE WORD OF ZEROES ON THE STACK
+      PUSHA   0,4(,15)           POINT TO PLIST WITH STACK TOP WORD
+      SVC     24   (READFDR)     ISSUE SVC

LAB    READFDR LIBRARY='USERLIB',FILE=(R1),VOLUME=SYSVOL,AREA=(R6)
+ LAB  PUSHN   0,36               GET SPACE ON STACK FOR PLIST
+      MVC     0(8,15),=CL8'USERLIB'   SET LIBRARY NAME
+      MVC     8(8,15),0(R1)      SET FILE NAME
+      MVC     16(6,15),SYSVOL    SET VOLUME NAME
+      MVI     22(15),X'00'       CLEAR FLAGS
+      MVI     23(15),0           INDICATE READ FDR1 ONLY
+      ST      R6,24(,15)         SET FDR RECEIVING AREA ADDRESS
+      XC      28(8,15),28(15)    (THIS FIELD RESERVED)
+      PUSHA   0,0                GET ONE WORD OF ZEROES ON THE STACK
+      PUSHA   0,4(,15)           POINT TO PLIST WITH STACK TOP WORD
+      SVC     24   (READFDR)     ISSUE SVC
```

Read Volume Table of Contents

Syntax

```
                                {ATTRIBUTES}
                                {EXTENTS   }
[label]   READVTOC   OPTION =   {LIBRARIES }
                                {FILES     }
                                {BLOCKS    }

                    [,PLIST=    {(register) }]
                    [           { expression}]

                    [,VOLUME=   {(register) }]
                    [           { expression}]
                    [           {'literal'  }]

                    [,LIBRARY=  {(register) }]
                    [           { expression}]
                    [           {'literal'  }]

                    [,COUNT=    {(register)}]
                    [           { number   }]

                    [,START=    {(register) }]
                    [           { expression}]
                    [           {    1      }]

                    [,OFB=      {(register) }]
                    [           { expression}]
```

Restrictions

The area addressed by PLIST must be in the user's Segment 2. If any operands are supplied as 'literals' (and in some other cases), the user must allow for generation of a literal pool.

Function

Provides information from a Disk Volume Table of Contents (VTOC). Specific functions are described under OPTIONS.

READVTOC issues a return code in the stack top word, indicating success, or the reason for failure, of the operation.

If PLIST is not specified, space for the parameter list is obtained from the stack; the length of the area is returned in General Register 1 (whose previous contents are lost).

If PLIST is specified, the designated area must be large enough to hold the desired output.

Operand Descriptions

OPTION= One of the following, coded as shown, indicating which type of information is desired. This operand is required, unless PLIST is specified.

ATTRIBUTES - 1. VTOC extents in use.
Number of unused blocks in VTOC.

2. Number of Libraries on Volume.
Number of Files on Volume.

3. Number of Free Extents on Volume.
Total size of Free Extents.

4. Descriptions of m (m=COUNT) largest Free Extents from nth (n=START) Free Extent.

EXTENTS - Descriptions of m (m=COUNT) Free Extents from nth (n=START) Free Extent.

LIBRARIES - Lists m (m=COUNT) Library names and number of Files in each Library listed, starting from nth (n=START) Library name.

FILES - Lists m (m=COUNT) Filenames starting from nth (n=START) File in specified Library.

BLOCKS - Reads m (m=COUNT) consecutive VTOC blocks starting from nth (n=START) block in VTOC and copies into File specified by OFB=.

PLIST= An expression, or a register in parentheses, pointing to an area to be used as the READVTOC parameter list. If PLIST is specified, no OPTION is required, nor are any of the other operands (in this case, it is assumed that the user has placed values in the PLIST for operands that would otherwise have been required).

VOLUME= An expression, a register in parentheses pointing to a 6-byte name, or a literal in single quotes, indicating the Volume from which VTOC information is desired. Required for all OPTIONs (unless PLIST is specified).

LIBRARY= An expression, a register in parentheses pointing to an 8-byte name, or a literal in single quotes indicating the Library about which VTOC information is desired. Required when OPTION=FILES (unless PLIST is specified).

COUNT= A number or a register in parentheses containing a number, indicating how many items (one or more) are requested (see OPTION descriptions). Required for all OPTIONs unless "PLIST" is specified.

START= An expression, or a register in parentheses containing a number, indicating which item (see OPTION descriptions) is the first item requested. Required for all OPTIONs (unless "PLIST" is specified). START=1 is the default.

OFB= The address, or a register in parentheses containing the address, of the Open File Block. The file specified must be open for output with enough space allocated to accommodate m VTOC blocks (as specified in BLOCKS).

Return Codes

0 - Requested operation performed.
4 - Invalid argument PLIST address.
8 - VOLUME not mounted.
12 - VOLUME used exclusively by another user or job.
16 - Insufficient buffer space to perform operation.
20 - Invalid OPTION request.
24 - LIBRARY not found.
28 - VTOC error; FDX1 and FDX2 conflict.
32 - Disk I/O error; VTOC not reliable.

Examples

```
 FOOBAR READVTOC OPTION=ATTRIBUTES,VOLUME='VOLVO',COUNT=32,START=(R8)
+FOOBAR  DS    0H
+        LA    1,222                SIZE OF PARAMETER LIST
+        PUSHN 0,0(,1)              SPACE FOR PARAMETER LIST
+        MVC   8(2,15),=Y(32)       SET COUNT FIELD
+        MVI   6(15),0              INSERT OPTION BYTE
+        STH   R8,10(15)            SET START FIELD
+        MVC   0(6,15),=CL6'VOLVO'  MOVE IN VOLUME NAME
+        PUSH  0,15                 PARAMETER LIST TO STACK
+        SVC   19 (READVTOC)        ISSUE READVTOC SVC
```

```
         READVTOC OPTION=EXTENTS,VOLUME=VCBSER,COUNT=(7)
+        DS       0H
+        LR       1,7                  COPY COUNT
+        MH       1,=Y(6)              TIMES ELEMENT SIZE
+        LA       1,4(,1)              PLUS MINIMUM SECTION LENGTH
+        PUSHN    0,0(,1)              GET SPACE REQUIRED
+        STH      7,8(15)              SET COUNT FIELD
+        MVI      6(15),1              INSERT OPTION BYTE
+        MVC      10(2,15),=Y(1)       SET START FIELD
+        MVC      0(6,15),VCBSER       MOVE IN VOLUME NAME
+        PUSH     0,15                 PARAMETER ADDRESS TO STACK
+        SVC      19 (READVTOC)        ISSUE READVTOC SVC

         READVTOC OPTION=LIBRARIES,VOLUME=(6),COUNT=32
+        DS       0H
+        LA       1,340                SIZE OF PARAMETER LIST
+        PUSHN    0,0(,1)              SPACE FOR PARAMETER LIST
+        MVC      8(2,15),=Y(32)       SET START FIELD
+        MVI      6(15),2              INSERT OPTION BYTE
+        MVC      10(2,15),=Y(1)       SET START FIELD
+        MVC      0(6,15),0(6)         MOVE IN VOLUME NAME
+        PUSH     0,15                 PARAMETER LIST TO STACK
+        SVC      19 (READVTOC)        ISSUE READVTOC SVC

         READVTOC OPTION=FILES,VOLUME='SYSTEM',LIBRARY='SYSS',COUNT=16
+        DS       0H
+        LA       1,148                SIZE OF PARAMETER LIST
+        PUSHN    0,0(,1)              SPACE FOR PARAMETER LIST
+        MVC      0(2,15),=Y(16)       SET COUNT FIELD
+        MVI      6(15),3              INSERT OPTION BYTE
+        MVC      10(2,15),=Y(1)       SET START FIELD
+        MVC      0(6,15),=CL6'SYSTEM' MOVE IN VOLUME NAME
+        MVC      12(8,15),=CL8'SYSS'  MOVE IN LIBRARY NAME
+        PUSH     0,15                 PARAMETER LIST TO STACK
+        SVC      19 (READVTOC)        ISSUE READVTOC SVC

         READVTOC OPTION=BLOCKS,VOLUME=VCBSER,COUNT=(3),START=(4),
                  OFB=(ROFB)
+        DS       0H
+        LA       1,20                 SIZE OF PARAMETER LIST
+        PUSHN    0,0(,1)              SPACE FOR PARAMETER LIST
+        STH      3,8(15)              SET COUNT FIELD
+        MVI      6(15),4              INSERT OPTION BYTE
+        STH      4,10(15)             SET START FIELD
+        MVC      0(6,15),VCBSER       MOVE IN VOLUME NAME
+        ST       ROFB,12(,15)         SET OFB ADDRESS
+        PUSH     0,15                 PARAMETER LIST TO STACK
+        SVC      19 (READVTOC)        ISSUE READVTOC SVC

         READVTOC PLIST=(RLIST),START=
+        DS       0H
+        PUSH     0,RLIST              PARAMETER ADDRESS TO STACK
+        SVC      19 (READVTOC)        ISSUE READVTOC SVC
```

Register Equation

Syntax

```
REGS     FP = {YES}
              {NO }
```

Restrictions

None.

Function

The REGS macroinstruction equates register numbers with the standard symbolic names used by all other system macroinstructions which refer to general registers. It should be included in all program assemblies which make use of system macroinstructions. Register names are as follows:

General Register Numbers	Names
0	R0
1	R1,AP
2	R2
3	R3
4	R4
5	R5
6	R6
7	R7
8	R8
9	R9
10	R10
11	R11
12	R12
13	R13,EP
14	R14
15	R15,SP

Floating Point Register Numbers	Names
0	F0
2	F2
4	F4
6	F6

Operand Descriptions

FP= If NO is specified, symbolic names for the Floating-Point Registers are not generated. The default is to YES.

Rename a Disk File

Syntax

```
1.  [label]  RENAME  PLIST=  { address  }
                             {(register)}

2.  [label]  RENAME  LIBRARY                ,LIBRARY={ address }
                                                     {'string' }

                   ,VOLUME={ address  }      ,NEWNAME={ address }
                           {'string'  }               {'string' }

                 [,RESTRICT={NO }]         [,BYPASS={NO }]
                            {YES}                    {YES}

3.  [label]  RENAME  FILE={ address }    ,LIBRARY={ address }
                          {'string' }             {'string' }

                   ,VOLUME={ address  }      ,NEWNAME={ address }
                           {'string'  }               {'string' }

        [,NEWLIB ={ address }] [,RESTRICT={NO }] [,BYPASS={NO }]
                  {'string' }             {YES}           {YES}
```

Restrictions

If any of the operands is specified as a character string in single quotes, then the issuing program must provide for the generation of a literal pool.

RENAME now examines all of the bits of the Option Byte in the input parameter list. Previously, bits 3-7 were not examined. Therefore, previously coded invocations of RENAME may fail or produce undesirable results if bits 3-7 are set. Bits 5-7 of the Option Byte are reserved and must be zeroes.

RENAME requires a minimum of 2K bytes of stack for buffer space to rename a library or a file. RENAME requires a minimum of 9K of stack for buffer space to rename both a library name and a file name for a given file ("Full RENAME").

Function

To rename a disk file or a library on a volume. A "Full RENAME" (renaming both a library name and a file name for a given file) may alter the Volume Table of Contents; otherwise, the structure of the VTOC is not altered. No file that is to be renamed may be open when the RENAME is attempted. Note that a "Full RENAME" is equivalent to moving a file from one library to another on the same volume.

If the PLIST= option is not utilized, then RENAME dynamically builds its parameter list on the stack, and it becomes the invoking program's responsibility to pop 32 bytes (40 for "Full RENAME") off the stack beyond the return code word.

Operand Descriptions

PLIST= The address of a user-generated parameter list, the format of which is described in the RENAME SVC in Chapter 6. If PLIST= is specified, no other operand is allowed; if it is not specified, the macro generates code to dynamically build a parameter list on the stack prior to issuance of the RENAME SVC.

PLIST may be specified as a register in parentheses containing the address of the user-generated parameter list, or as an expression addressing it.

LIBRARY Indicates that the library specified in the LIBRARY= operand is to be renamed. Use of this operand is mutually exclusive with the FILE= operand. Note that this operation is equivalent to "moving" all the files in that library to a new library on the same volume. Libraries may not, however, be merged in this manner: the library specified by the NEWNAME= operand can exist when the RENAME SVC is issued.

FILE= Specifies the name of the file to be RENAMEd. This operand may be specified as a character string in single quotes which is the name of the file, or as an address expression containing the name of the file to be renamed. Use of this operand is mutually exclusive with the LIBRARY operand described above.

LIBRARY= Specifies the name of the library to be RENAMEd or the name of the library containing the file to be renamed. This operand may be specified as a character string in single quotes which is the name of the library, or as an address expression pointing to an eight-byte field containing the library name. This operand is required if PLIST= is not specified.

VOLUME= Specifies the name of the volume containing the file and/or library to be RENAMEd. This operand may be specified as a character string in single quotes which is the name of the volume, or as an address expression pointing to an six-byte field containing the volume name. This operand is required if PLIST= is not specified.

NEWNAME= Specifies the new name of the file or library being RENAMEd. This operand may be specified as a character string in single quotes which is the name of the library or file, or as an address expression pointing to an eight-byte field containing the library or file name. This operand is required if PLIST= is not specified.

RESTRICT= YES specifies that the RENAME SVC is to ignore any special access rights which may have been granted to the invoking program, and thus restrict itself to the user's LOGON access rights in determining whether the user may RENAME the specified file(s). If NO is specified, or the operand is omitted, the RENAME operation proceeds, utilizing current file access rights.

BYPASS= If NO is specified, or the operand is omitted, the RENAME operation performs an expiration date check. If the date is unexpired, the entire RENAME operation is not performed. If YES is specified, the expiration date check is bypassed.

NEWLIB= The name of the library in which the renamed file is to be placed. This operand is used only for the "Full RENAME" option, that is, for the renaming of both the file name and the library name for a given file. The library specified by the NEWLIB= operand can exist when the RENAME SVC is issued. If omitted, then the same library as specified by the LIBRARY= operand is assumed. This operand may be specified as a character string in single quotes which is the new file name or library name, or as an address expression pointing to an eight-byte field containing the new file name or library name. Use of this operand is mutually exclusive with use of the LIBRARY operand.

Example

```
LAB1      RENAME
FILE=RENFILE,LIBRARY=RENLIBR,VOLUME=RENVOLUME,     X
               NEWNAME=RENNEWNAME
+LAB1     PUSHN  0,8
+         MVI    0(15),0
+         MVI    6(15),B'00000000'
+         MVC    0(6,15),RENVOLUME          VOLUME
+         PUSHC  0(8),RENNEWNAME            NEW NAME
+         PUSHC  0(8),RENFILE               FILE
+         PUSHC  0(8),RENLIBR               LIBRARY
+         SVC    26 (RENAME)
```

```
LAB    RENAME  PLIST=(R1)
+LAB   PUSH    0,R1     POINT TO USER-DEFINED PARAMETER LIST
+      SVC     26       (RENAME)


LAB    RENAME  LIBRARY,LIBRARY=OLDLIB,VOLUME='MYVOL',NEWNAME=NEWLIB
+LAB   PUSHN   0,8            GET TWO WORDS ON THE STACK
+      MVI     7(15),0        RESERVED; MUST BE ZERO
+      MVI     6(15),B'01000000'       SET OPTION FLAGS
+      MVC     0(6,15),=CL6'MYVOL'     CURRENT VOLUME NAME
+      PUSHC   0(8),NEWLIB             NEW FILE/LIBRARY NAME
       PUSHN   0,8            RENAME LIBRARY (FILENAME OMITTED)
       PUSHC   0,8,OLDLIB     CURRENT LIBRARY NAME
       PUSH    0,15           POINT TO PLIST WITH STACK TOP WORD
       SVC     26       (RENAME)
```

Remove Timer Interval

Syntax

```
[label]  RESETIME
```

Function

Cancels an interval timing request previously established by 'SETIME' which has not been the subject of a 'CHECK INTERVAL' or previous 'RESETIME'. A programming error is assumed and the issuing program cancelled if there is no such request.

Operand Description

There are no operands.

Example

```
 LAB1     RESETIME
+LAB1     PUSHN      0,4
+         MVI        0(15),X'80' RESET
+         SVC        32 (RESETIME)
```

Return to Invoker

Syntax

```
                                [                   ] [             ]
                                |,CODE={(register)}| |,COND=number|
[label] RETURN [UNLINK]         |      {expression}| |             |
                                [                   ] [             ]
```

Restrictions

(A CALL, LINK or program invocation must have occurred for the issuing task.)

Function

The RETURN macroinstruction is used to (conditionally) exit from a program to the system when normal termination of the run is required. It is also used to exit from a subprogram and return to the calling program. The stack top pointer (register 15) and control register 1 are restored to their values before the CALL or LINK which effected entry to the program or subprogram. General register 1-14 contents are restored to their state before the CALL, LINK or program invocation. A return code, if requested, is set in register 0. Otherwise, register 0 is set to zero. (Note that "RETURN CODE=(0)" leaves register 0 unchanged.)

Operand Descriptions

UNLINK — Specifies return to the most recent LINK issuer, Command Processor or Procedure Interpreter, thus terminating all routines invoked by a sequence of CALLs. 'COND=' must not be specified with this operand.

CODE= — If the CODE= operand is supplied, register 0 is loaded with the number specified or from the register specified. In this case, the instruction "LA 0,number" or "LR 0,Rn" is generated.

COND= — If supplied, specifies the condition codes under which the return is to be made, as for a machine instruction. If omitted, 'COND=15' is assumed. Invalid if 'UNLINK' operand specified.

Example

```
 LAB1     RETURN     CODE=ZERO,COND=7
+LAB1     LA         R0,ZERO
+         RTC        7
```

Rewrite a Record

Syntax

```
[label]  REWRITE  [TABS,     ]UFB={(register)}[,COND=number]
                  [SELECTED,]    {expression}
```

Function

Rewrites a disk record or workstation line. The file must be open in IO or SHARED mode or placed in temporary IO mode by the 'START IO' function. In IO mode, the last successful function addressed to the file must have been a READ with HOLD option unless the file is a workstation file. In SHARED mode, the program must be HOLDing the record to be rewritten (as a result of a preceding READ with the HOLD option not overridden by a later READ with HOLD. Record or line is taken from the user's record area as addressed by field UFBRECAREA of the specified User File Block (UFB).

Additional control information (order area) precedes the line to be written in the record area for workstation line REWRITEs. Refer to the specific device description for details on this area.

For indexed disk file REWRITEs, the key field in the record to be rewritten is validated. REWRITE may not change this field.

An error condition discovered on REWRITE will result in nonzero ASCII digit settings of the file status bytes (UFBFS1, UFBFS2) and return to the address in UFBEODAD or UFBERRAD, with the normal return address in register 0.

Possible file status codes indicating errors are:

Return to UFBEODAD:

23 - Block beyond end of file for block-level I/O

Return to UFBERRAD:

30 - Permanent I/O error
34 - Order check on workstation
95 - Invalid function or function sequence (includes key validation failure for indexed file REWRITE)

If UFBERRAD is binary zeroes, these conditions cause program termination.

Operand Descriptions

TABS — Indicates that bytes 4-13 of the user's record area contain tab position settings for the workstation (in ascending order, terminated by the first zero item, binary column numbers 1-80), and that the purpose of the REWRITE is to set these tabs.

SELECTED — Indicates that only those fields with selected-field tags set in their field attribute characters are to be written to a workstation screen.

UFB= — The address of a User File Block (UFB), which may be supplied as a register specification in parentheses, where the register contains the UFB address, or as an expression not in parentheses, where the word addressed is assumed to begin the UFB.

COND= — If specified, the number or absolute expression becomes the first operand of the JSCI instruction by which the REWRITE function is entered. Thus the REWRITE is made conditional. COND=15 is the default. Register 1 is loaded with the UFB address even when the condition is not satisfied.

Example

```
 LAB1     REWRITE    UFB=(R2)
+LAB1     LR         1,R2                 SET REGISTER 1
+         MVI        8(1),B'00000000'     MODIFIERS
+         JSCI       15,8(,1)             REWRITE FUNCTION
```

Scratch a File

Syntax

```
1.   [label]  SCRATCH  PLIST=   {expression}
                                {(register)}

2.   [label]  SCRATCH  {LIBRARY          } ,LIBRARY={expression}
                       {FILE={expression}}          {'literal' }
                       {     {'literal' }}
                                        ,VOLUME={expression}
                                                {'literal' }

                                       [,RESTRICT={ NO}]
                                       [          {YES}]

                                       [  ,BYPASS={ NO}]
                                       [          {YES}]
```

Function

To delete ("scratch") a disk file or a library of disk files on a volume, making the space utilized by the file(s) available for reallocation and removing all references to the file(s) from the Volume Table Of Contents (VTOC). No file that is to be deleted may be open when the SCRATCH is attempted.

Return codes in binary in the top word of the stack indicate the result of the request:

```
Return code =  0 - File or library successfully
                   scratched
Return code =  4 - Volume not mounted
Return code =  8 - Volume used exclusively by other user
Return code = 12 - All buffers in use, no scratch
Return code = 16 - Library not found
Return code = 20 - File not found
Return code = 24 - Update access denied, no scratch
                   (single-file scratch only)
Return code = 28 - Unexpired file, no scratch
                   (single-file scratch only)
Return code = 32 - File in use, no scratch
Return code = 36 - VTOC error! FDX1 and FDX2 don't
                   agree
Return code = 40 - VTOC error! FDX2 and FDR don't agree
Return code = 44 - Invalid argument list address
Return code = 48 - I/O error! VTOC unreliable!
Return code = 52 - Open, protected, and/or unexpired
                   file bypassed in scratching library
```

If space on the volume is lost during SCRATCH because there is no room in the VTOC to record released extents, the high-order three bytes of the return code word contain the number of blocks lost. Otherwise they are zeroed.

Note: If the PLIST= option is not utilized, it is the program's responsibility to pop 24 bytes off the stack beyond the return code word on the stack.

Operand Descriptions

PLIST= - The address of a SCRATCH parameter list. (For a description of the format of the parameter list, please refer to the description of the SCRATCH SVC in Chapter 6.) If PLIST= is specified, no other operand may be specified. If PLIST= is not specified, the macro generates code to dynamically build a parameter list on the stack prior to issuance of the SCRATCH SVC.

LIBRARY - Indicates that all files within the library specified are to be deleted. Use of this operand is mutually exclusive with the FILE= operand.

FILE= - Specifies the address at which the file's name is located. May be specified as a character string delimited by single quotes, in which case a constant is assumed. Use of this operand is mutually exclusive with the LIBRARY operand described above.

LIBRARY= - Specifies the address at which the library's name is located. May be specified as a character string delimited by single quotes, in which case a constant is assumed. This operand is required if PLIST= is not specified.

VOLUME= - Specifies the address at which the volume's name is located. May be specified as a character string delimited by single quotes, in which case a constant is assumed. This operand is required if PLIST= is not specified.

RESTRICT= - If NO is specified, or the operand is omitted, the SCRATCH operation proceeds utilizing current file access rights. If YES is specified, the operation is restricted, assuming only the file access rights of the user and ignoring any special access rights of the program.

BYPASS= - If NO is specified, or the operand is omitted, the SCRATCH operation performs an expiration date check. For any unexpired file(s), the SCRATCH is not performed. If YES is specified, the expiration date check is bypassed.

Example

```
 LAB1    SCRATCH FILE=SCRFILE,LIBRARY=SCRLIBR,VOLUME=SCRVOLUME
+LAB1    PUSHN   0,8
+        MVI     0(15),0
+        MVI     6(15),B'00000000'
+        MVC     0 (6,15),SCRVOLUME          VOLUME
+        PUSHC   0(8),SCRFILE                FILE
+        PUSHC   0(8),SCRLIBR                LIBRARY
+        SVC     27 (SCRATCH)
```

Set Task-Related Defaults

Syntax

```
[label]  SET PROGVOL= {(register)}        ,PROGLIB= {(register)}
                      {'string'  }                  {'string'  }
                      {address   }                  {address   }

          ,INVOL= {(register)}          ,INLIB= {(register)}
                  {'string'  }                  {'string'  }
                  {address   }                  {address   }

         ,OUTVOL= {(register)}         ,OUTLIB= {(register)}
                  {'string'  }                  {'string'  }
                  {address   }                  { address  }

        ,POOLVOL= {(register)}        ,WORKVOL= {(register)}
                  {'string'  }                  {'string'  }
                  {address   }                  {address   }

        ,PRINTER= {(register)}      ,PRNTMODE= {(register)}
                  {'string'  }                  {'string'  }
                  {address   }                  {address   }

     ,FILECLASS= {(register)}          ,LINES= {(register)}
                  {'string'  }                  {'string'  }
                  {address   }                  {address   }

      ,PRTCLASS= {(register)}          ,FORM#= {(register)}
                  {'string'  }                  {'string'  }
                  {address   }                  {address   }

         ,RUNVOL= {(register)}         ,RUNLIB= {(register)}
                  {'string'  }                  {'string'  }
                  {address   }                  {address   }

      ,JOBQUEUE= {(register)}       ,JOBCLASS= {(register)}
                  {'string'  }                  {'string'  }
                  {address   }                  {address   }

     ,JOBLIMIT= {(register)}
                {'string'  }
                {address   }
```

Restrictions

All library and volume name specifications (except literals) must reference eight- and six-byte fields respectively, as the SET SVC cannot determine the length of the character string and assumes the maximum.

If any operand is specified as a literal or an integer, then the user program must allow for the generation of a literal pool.

Function

Allows user programs to set default values according to the operands specified. These values are used by the various system utilities and SVCs. Note that none of the operands have defaults and that any unspecified operands will be unaffected.

Operand Description

NOTES:

(1) All operands are optional (although at least one should be specified).

(2) Operands may be specified as:

(a) A register in parentheses pointing to a character string which is the desired value. If the item is numeric (PRINTER, LINES, or FORM#), then the value is assumed to be in binary.

(b) A character string in single quotes which is the desired value, except for the numeric items (PRINTER, FORM#, and LINES) which use an integer (not in quotes) which is the desired value in decimal,

(c) An expression addressing a character string which is the desired value. If the item is numeric (PRINTER, LINES, or FORM#), then the value is assumed to be in binary.

PROGVOL= Default program/procedure volume name.

PROGLIB= Default program/procedure library name. This pair of operands is used only in procedures, for programs run by those procedures, and identify default library and volume names for all such programs.

INVOL=
INLIB=
Default non-output volume name.
Default non-output library name. This pair of operands is used primarily by the OPEN SVC to locate files OPENed as input.

OUTVOL=
OUTLIB=
Default output volume name.
Default output library name. This pair of operands is used primarily by the OPEN SVC to assign files OPENed as output.

SPOOLVOL=
Default volume for assignment of SPOOLed (Print) files.

WORKVOL=
Default volume for assignment of WORK files.

PRINTER=
Default printer device number for on-line printing. Note that this operand in no way affects printer assignment for SPOOLed files. This number must be in the range 0 to 255.

PRNTMODE=
Default print mode. Permissible values and their meanings are as follows:

- 0 — ONLINE: Printing will be done using the printer as a direct output device; a Print file is NOT created.
- S — SPOOL: Print files will be created and will be queued by the System Task (@SYSTSK@) for printing at the earliest opportunity,
- K — KEEP: Print files will be created but will NOT be queued for printing by the System Task.
- H — HOLD: Print files will be placed in the user's Print Library and will be queued by the System Task, but will NOT be printed until requested by the system operator or the user.

FILECLASS=
Default file protection class. The following values are valid:

- # — Accessible only by Security Administrators and the Owner-of-Record.
- $ — READ only files. READ access granted to all users regardless of the individual access privileges.

	@	EXECUTE only files. EXECUTE access granted to all users as above.
	A-Z	Accessible by users with class access privileges matching the type of access desired.
	(BLANK)	Unprotected file. WRITE access implied for all users regardless of their individual access privileges.
LINES=		Default number of lines-per-page. This operand is used primarily by the Print functions of system utilities. This number must be in the range 0 to 255.
PRTCLASS=		Default print class. This operand determines the class to which print requests sent to the system task will be assigned. Printer assignment, scheduling priority, and header page options are set for each class by the system operator and, as such, may vary from time to time. Valid values are the letters A-Z, or a blank.
FORM#=		Default form number for Print files. The association of a form number with a specified form is installation-defined. Thia number becomes part of the queue record for a Print file and is examined by the System Task. This number must be in the range 0 to 254.
RUNVOL= RUNLIB=		Default program/procedure execution volume and library name. This operand pair is used by the Command Processor RUN command to locate program/procedures to be executed.
JOBQUEUE=		Default job status for a background job. Determines when the submitted background job is executed. Possible values are:
	R - Run:	The job is executed as soon as possible.
	H - Hold:	The job is held in the job queue until it is released for execution
JOBCLASS=		Default job class for a background job. Background jobs are processed according to the job class priority hierarchy specified from the Operator's Console. Within a given job class, background jobs are processed in order of submittal. Possible values are A-Z.

JOBLIMIT= Default CPU time limit for job execution. The time limit is specified in seconds. Possible values are 0-35999 (thus the maximum time limit is 99:59:59). If zero, then the job has no time limit.

Example

```
LAB       SET     PROGVOL=(R2),PROGLIB='MYLIB',PRINTER=PRTID,FORM#=(R5),LINES=55
+LAB      PUSH    0,0                 SAVE REGISTER ZERO IN THE STACK
+         PUSHN   0,64                PUSH AREA FOR SVC PLIST
+         XC      0(64,15),0(15)      INITIALIZE AREA TO ZEROES
+*
+*                 SET DEFAULT PROGRAM VOLUME NAME
+         ST      R2,0(,15)           PLACE ADDRESS IN PLIST
+*
+*                 SET DEFAULT PROGRAM LIBRARY NAME
+         LA      R0,=CL8'MYLIB'      POINT TO LITERAL
+         ST      R0,4(,15)           SET ADDRESS IN PLIST
+*
+*                 SET DEFAULT PRINTER NUMBER
+         LA      R0,PRTID            POINT TO DATA ITEM
+         ST      R0,40(,15)          PLACE ADDRESS IN PLIST
+*
+*                 SET DEFAULT LINES-PER-PAGE
+         LA      R0,=AL1(55)         POINT TO LITERAL
+         ST      R0,52(,15)          PLACE ADDRESS IN PLIST
+*
+*                 SET DEFAULT FORM NUMBER
+         ST      R5,60(,15)          PLACE ADDRESS IN PLIST
+         OI      60(15),X'80'        FLAG END OF PLIST
+         SVC     35 (SET)            ISSUE SVC
+         POP     0,0                 RESTORE REGISTER ZERO FROM STACK
```

Set Interval Timer

Syntax

```
[label] SETIME {UNTIL}= {(register)}
               {CSEC }  {expression}
```

Function

Sets a timer interval for the issuing task to expire at the time specified, or after the number of 1/100 second units specified. If a previous interval timing request was active for this task, it is cancelled and the new one instated.

Operand Descriptions

UNTIL= Either a register specification in parentheses, where the register contains a binary time value in 1/100 second units into a day (from midnight), or an address expression, where the four bytes starting at that address contain the time as above. To request expiration at some time tomorrow, the value supplied must be 24 hours plus the required time-of-day. A requested time less than the current time-of-day will result in immediate expiration.

CSEC= Either a register specification in parentheses, where the register contains the number of 1/100 second units to delay processing, in binary; or an expression, not in parentheses, for the required number of 1/100 second units. May not exceed one day.

Example

```
 LAB1     SETIME   CSEC=55
+LAB1     PUSHA    0,55
+         MVI      0(15),0     UNITS
+         SVC      32 (SETIME)
```

Start File Processing in Specified Mode or at Specified Record Location

Syntax

```
[label]  START  {IO      },UFB={(register)}[,COND=number]
                {OUTPUT  }     {expression}
                {EXTEND  }
                {BEGIN   }
                {SKIP    }
                {EQ      }
                {GT      }
                {GE      }
                {ATTNT   }
                {WAIT    }
                {HOLD    }
                {RELEASE}
                {TCWAIT[,MULTIPLE      ][TIMEOUT={register  }]}
                {      [(MULTIPLE,ATTN)][         {expression}]}
                {HALTIO}
```

Function

The function of START differs for various file types.

(1) Consecutive disk files (normal DMS):

START is valid in OUTPUT or EXTEND open modes only. 'START IO' writes any remaining buffered records to disk, and then enters temporary IO mode, with the next record to be read set to the first record of the file. 'START OUTPUT' places the file in OUTPUT mode, after effectively deleting all records in the file (but not necessarily releasing space allocated for them on a disk file). The next WRITE will then place a new first record in the file. 'START EXTEND' places the file in EXTEND mode (thus having significant effect only when 'START IO' has been previously issued). The next WRITE will then add a record to the end of the file. Possible error indications in the file status bytes (UFBFS1, UFBFS2) are as follows:

30 - Permanent I/O error
95 - Invalid function or function sequence

(2) Consecutive disk files:
(variable-length records, normal DMS):

START BEGIN and START SKIP are valid in INPUT and IO modes. A READ NEXT issued after START BEGIN will read the first record of the file. A READ NEXT issued after a START SKIP (with a signed binary number "n" in the word addressed by UFBKEYAREA) will skip over "n" records and read the record after them (n>0), will merely read the next record (n=0), will reread the current record (n=-1) or will read a preceding record (n<-1).

(3) Consecutive disk and magnetic tape files (physical access method):

START WAIT is valid in INPUT, OUTPUT, or IO modes. The program is paused until a preceding READ or WRITE operation is completed. START IO and START OUTPUT have the same function as for normal consecutive DMS. Possible error indications in the file status bytes (UFBFS1, UFBFS2) are as follows:

30 - Permanent I/O error
95 - Invalid function or function sequence (including START WAIT issued without preceding block-level READ, REWRITE OR WRITE)

(4) Indexed disk files:

START is valid in INPUT, IO or SHARED modes only. Valid options are EQ, GT and GE. The START function is essentially a READ (KEYED, NODATA) operation (key from area addressed by UFBKEYAREA, with length UFBKEYSIZE) with the following additional options:

EQ - If a record with the specified key is not found in the file, invalid-key and no-record-found conditions are indicated. (This is like READ KEYED.)

GT - The first record with key greater than the supplied key is sought. (Collating sequence is normal ASCII.) If no such record is found, invalid-key and boundary-violation conditions are indicated.

GE - The first record with key greater than or equal to the supplied key is sought. Otherwise like the GT option.

After a successful START function, a succeeding READ (without KEYED option) will read the record located by START. Successive READs will then read successive records.

If UFBGKSIZE is not all binary zeroes, the binary value in UFBGKSIZE is used as the key length for the above searches, in place of UFBKEYSIZE. UFBGKSIZE may be set by the user's program before issuing a START. It must always be less than or equal to UFBKEYSIZE. If not, a fatal error resulting in program termination will occur. UFBGKSIZE is set to zero by every such START function.

Possible invalid-key and error indications in the file status bytes (UFBFS1, UFBFS2) are as follows:

23 - Invalid-key, no record found
24 - Invalid-key, boundary violation
30 - Permanent I/O error
95 - Invalid function or function sequence

(5) Workstation files:

The only valid option is 'ATTNT'. Only the file status bytes are modified. They are set as follows:

UFBFS1 - 0
UFBFS2 - AID character as indicated on the most recent interruption for this workstation; hexadecimal values as follows:

20 - Keyboard unlocked.
21 - Keyboard locked by REWRITE function or other write to workstation.
3F - Display screen, tab positions, or other workstation status lost.
Other - Indication of last AID character (e.g., ENTER, PROGRAM FUNCTION) received. See specific device description.

(6) Disk files (IO or SHARED open modes only):

START HOLD acquires temporary exclusive control of the entire File addressed. It has no significant effect in IO mode.

START RELEASE may be used to remove a record or File from HOLD status without issuing a REWRITE, DELETE, or another READ with the HOLD option. It has no significant effect in IO mode.

For all START functions and all file types, an invalid-key condition results in return to the address in UFBEODAD, with the normal return point address in register 0. Other exceptional and error conditions result in return to the address in UFBERRAD, with the normal return point address in register 0. If UFBEODAD is zero, UFBERRAD is used in its place. If UFBERRAD is zero as well, any exceptional condition results in abnormal termination of the program.

(7) Telecommunication devices:

START TCWAIT waits for the completion of current 'READ' or 'WRITE' operations issued on this TC file (this UFB).

START TCWAIT, MULTIPLE waits for completions on all TC devices for which this program has an outstanding 'READ' or 'WRITE' operation.

START,TCWAIT,(MULTIPLE,ATTN) waits for unsolicited interrupts for any TC lines, which this program controls, in addition to START TCWAIT, MULTIPLE.

The TIMEOUT operand can be used in conjunction with either of the above options. The expression field is an unsigned integer with value less than or equal to 255. If 'register' is specified, the right-most byte of the register will be used. In either case, TIMEOUT specifies the time interval in seconds.

The following table summarizes the uses of START:

START - MODES OF USE WITH DISK FILES

	Open for Input	Open for Output	Open for I/O	Open for EXTEND	Open for Shared IO
Fixed Consecutive RAM		Start IO OUTPUT EXTEND		Start IO OUTPUT EXTEND	
Variable Length Consecutive RAM	Start SKIP BEGIN	Start IO OUTPUT EXTEND	Start SKIP BEGIN	Start IO OUTPUT EXTEND	
Indexed RAM	Start EQ GT GE		Start EQ GT GE		Start EQ GT GE Start HOLD RELEASE
BAM		Start IO OUTPUT EXTEND		Start IO OUTPUT EXTEND	
PAM	Start WAIT	Start WAIT Start IO OUTPUT	Start WAIT		

Operand Descriptions

IO
OUTPUT
EXTEND
BEGIN
SKIP
EQ
GT
GE
ATTNT
WAIT
HOLD
RELEASE
TCWAIT
HALTIO

As described above.

UFB= The address of a User File Block (UFB), which may be presented as a register specification in parentheses, where the register contains the UFB address, or as an expression not in parentheses, where the word at the address designated is assumed to begin the UFB.

COND= If specified, the number or absolute expression becomes the first operand of the JSCI instruction by which the START function is entered. Thus the START is made conditional. COND=15 is the default. Register 1 is loaded with the UFB address even when the condition is not satisfied.

Example

```
 LAB1     START    GE,UFB=(R2)
+LAB1     LR       1,R2                  SET REGISTER 1
+         MVI      16(1),B'00000011'     MODIFIERS
+         JSCI     15,16(1)              START FUNCTION
```

[deleted]

[deleted]

Submit Job or Print Request

Syntax

```
(1) [label] SUBMIT JOB [,PLIST= {(register) }] [,PROCNAME={(register)}]
                       [        { expression}] [          {'literal' }]

    [,LIBRARY= {(register) }] [,VOLUME=  {(register) }]
    [          {'literal'  }] [          {'literal'  }]
    [          { expression}] [          { expression}]

    [,JOBNAME= {(register) }] [,JOBCLASS={(register) }]
    [          {'literal'  }] [          {'literal'  }]
    [          { expression}] [          { expression}]

    [,STATUS=  {'RUN'      }]    [,DISP= {'REQUEUE'  }]
    [          {'HOLD'     }]    [       { expression}]
    [          { expression}]    [       {           }]

    [,CPULIMIT=({(register) } [,{'CANCEL'    }])] [,DUMP={'YES'      }]
    [           { expression} [,{'PAUSE'     }] ] [      {'NO'       }]
    [                         [,{'WARN'      }] ] [      {'PROG'     }]
    [                         [,{ expression} ] ] [      { expression}]

(2) [label] SUBMIT PRINT[,PLIST={(register) }] [,FILENAME={(register) }]
                        [       { expression}] [          {'literal'  }]
                        [       {           }] [          { expression}]

              [,LIBRARY=  {(register) }] [,VOLUME=  {(register) }]
              [           {'literal'  }] [          {'literal'  }]
              [           { expression}] [          { expression}]

              [,PRTCLASS= {(register) }] [,FORM#=   {(register) }]
              [           {'literal'  }] [          {'literal'  }]
              [           { expression}] [          { expression}]

              [,COPIES=   {(register) }] [,STATUS=  {'SPOOL'    }]
              [           {'literal'  }] [          {'HOLD'     }]
              [           { expression}] [          { expression}]

              [,DISP=     {'REQUEUE'  }]
              [           {'SAVE'     }]
              [           { expression}]
```

Function

If initial parameter is JOB, SUBMIT requests the queuing of a procedure file for execution as a non-interactive job. If initial parameter is PRINT, SUBMIT requests the queuing of a print file for printing. SUBMIT issues a return code in the stack top word indicating the success/failure/status of the operation.

Operand Descriptions

PLIST= A 44-byte user-supplied parameter list (FULLWORD ALIGNED) for use by the SUBMIT SVC and constructed as follows:

For "JOB" Requests:

Bytes 0-7: The name of the Procedure (PROCNAME) to be run.

Bytes 8-15: The name of the LIBRARY in which the Procedure resides.

Bytes 16-21: The name of the VOLUME on which the Procedure resides.

Bytes 22-29: A user-supplied JOBNAME or spaces.

Byte 30: The JOBCLASS to which this job is to be queued.

Byte 31: The action to be taken in case of an abnormal termination of this job:

X'C0' - Produce a DUMP for this job ('YES').

X'80' - Do NOT produce a DUMP for this job ('NO').

X'00' - Produce a DUMP only if requested by the abnormally terminating program ('PROG').

Bytes 32-35: The CPU Time Limit (in timer units) imposed upon this job. If zero, then the job has no time limit.

Byte 36: The initial STATUS of this job when it is queued:

X'80' - HOLD - NOT eligible for scheduling until released by the operator or the submitter.

X'00' - RUN - eligible for scheduling upon submission of the request.

Byte 37: Whether or not to check for a CPU Time Limit, the action to be taken in case the limit is exceeded, and whether or not the job should be requeued after execution:

X'80' - Check for timer limit expiration (IF a CPU Time Limit is specified then this bit MUST be on).

X'40' - CANCEL this job if the CPU Time Limit is exceeded.

X'20' - PAUSE this job if the CPU Time Limit is exceeded. (If neither of these bits is on and a CPU Time Limit has been set, then a WARNing will be issued).

X'04' - REQUEUE this job after execution.

Bytes 38-43: RESERVED (Should be ZEROS).

For "PRINT" Requests:

Bytes 0-7: The name of the File (FILENAME) to be printed.

Bytes 8-15: The name of the LIBRARY in which the file resides.

Bytes 16-21: The name of the VOLUME on which the file resides.

Byte 22: The Print Class (PRTCLASS) to which this file is to be queued.

Byte 23: The Form Number (FORM#) (in binary) of this file to be printed.

Bytes 24-25: The number of COPIES (in binary) of this file to be printed.

Byte 26: The initial STATUS of this file when it is queued:

X'80' - HOLD - NOT eligible for printing until released by the operator or the submitter.

X'00' - SPOOL - eligible for printing upon submission of the request.

Byte 27: Whether or not this file should be REQUEUEd, SAVEed, or scratched after printing:

X'40' - REQUEUE this file after printing.

X'20' - SAVE this file after printing.

Bytes 28-43: RESERVED (Should be ZEROS).

"PLIST" may be specified either as a register in parentheses pointing to the user-supplied parameter list or as an expression addressing the user-supplied parameter list.

If "PLIST" is specified, then the remaining operands are optional and, if present, are used to modify the parameter list IN PLACE. The default values of any omitted operands are NOT recognized so as not to override the value set in the user's parameter list.

If "PLIST" is not specified, then the remaining operands are used to build a parameter list on the stack. The default values of omitted operands are used in this case. The user is responsible for POPping off the 44 bytes beyond the stack top word (SVC Return Code) on return.

"PROCNAME/FILENAME", "LIBRARY", "VOLUME", "JOBCLASS/ PRTCLASS", and "FORM#" are required by their respective functions unless "PLIST" is also specified. All other operands are always optional.

PROCNAME/FILENAME= The name of the Procedure to be run or the File to be printed.

LIBRARY= The name of the library in which the Procedure/File reside.

VOLUME= The name of the volume on which the Procedure/File reside.

JOBNAME= An optional user-supplied name for the job to be submitted (limited to 8 characters).

JOBCLASS/PRTCLASS= The class to which the job/print request is to be assigned. Valid values are the letters A-Z.

The above operands may be specified as a register in parentheses pointing to the required value, a literal in single quotes which is the required value, or an expression addressing a field containing the required value.

FORM#= The number of the form on which to print this file. This number must be in the range 0-254.

COPIES= The number of copies of this file to be printed. This number must be in the range 1-32767. The default value is 1.

"FORM#" and "COPIES" may be specified as a register in parentheses containing the value in binary, an integer not in quotes which is the value in decimal, or an expression addressing a field containing the value in binary.

CPULIMIT= The total amount of time that this job may use the CPU (1st sub-operand) and the action to be taken if that limit is exceeded (2nd sub-operand).

The actual time may be specified as a register in parentheses or an expression addressing a 4-byte field containing the limit in timer units. A value of zero implies that the job has no limit and any action indicated by the 2nd sub-operand will be ignored. The default is to zero (no limit).

The action to be taken upon completion may be specified either as one of the following literals in single quotes, or as an expression addressing a one-byte field containing the appropriate flag value (See "PLIST" entry for byte 37 ("JOB") above). The default is to "WARN".

'CANCEL' - Force abnormal termination of the Procedure.

'PAUSE' - Suspend execution of the Procedure until resumed by the operator.

'WARN' - Issue a WARNing message to the operator.

NOTE: The time (1st sub-operand) may always be specified by itself; the action on expiration (2nd sub-operand) may only be specified by itself if "PLIST" is also specified.

STATUS= The initial status of the request when it is placed on the queue. It may be specified either as one of the following literals in single quotes or as an expression addressing a one-byte field containing the appropriate flag value (see "PLIST" entry for byte 36 ("JOB") or byte 26 ("PRINT") above). The default is to 'RUN'/'SPOOL':

'RUN' - Eligible for scheduling upon submission of the request ("JOB" only).

'SPOOL' - Eligible for printing upon submission of the request ("PRINT" only).

'HOLD' - NOT eligible for print/execution scheduling until released by the operator or the submitter.

DISP= The action to be taken at completion of the request. It may be specified as a literal in single quotes or as an expression addressing a one-byte field containing the appropriate flag value (see "PLIST" entry for byte 37 ("JOB") or byte 27 ("PRINT") above). The default is to NOT set these options (do NOT requeue or save).

'REQUEUE' - Place the request back onto the queue for re-execution/re-printing (for "PRINT" requests, this implies 'SAVE').

'SAVE' - Do not scratch this file after printing ("PRINT" only).

DUMP= The action to be taken in the event of an abnormal termination. It may be specified as a literal in single quotes or as an expression addressing a one-byte field containing the appropriate flag value (see "PLIST" entry for byte 31 above). The default is to 'PROG'.

'YES' - Produce a DUMP for this job.

'NO' - Do NOT produce a DUMP for this job.

'PROG' - Produce a DUMP only if requested by the program abnormally terminating.

Return Codes

=	0	Successful
	4	Volume Not Mounted
	8	Volume in Exclusive Use
	12	All Buffers in Use, Unable to Perform Verification
	16	Library Not Found
	20	File Not Found
	24	Improper File Type (or Zero Records as Indicated in Label
	28	File Access Denied
	32	VTOC Error, FDX1/2 Do not Agree
	36	VTOC Error, FDX2/FDR Do not Agree
	40	Invalid Specification of File/Library/Volume
	44	VTOC Unreliable
	48	System Task not Running, NO SPOOLED PRINTING OR NON-INTERACTIVE JOBS
	52	Error in Performing XMIT to System Task
	56	Invalid Options Specified In Parameter List

Examples

```
LAB      SUBMIT JOB,PROCNAME='MYPROC',LIBRARY=PROCLIB,VOLUME=(R5),      X
                JOBCLASS='A',CPULIMIT=((R3),'PAUSE'),DISP='REQUEUE'
+LAB     PUSHN  0,44                    GET SPACE ON STACK FOR "PLIST"...
+        XC     0(44,15),0(15)          ... AND CLEAR IT TO ZEROES
+        MVC    0(8,15),*+10            SET PROCEDURE NAME
+        B      *+12                    BRANCH AROUND LITERAL
+        DC     CL8'MYPROC'             PROCEDURE NAME
+        MVC    8(8,15),PROCLIB         SET LIBRARY NAME
+        MVC    16(6,15),0(R5)          SET VOLUME NAME
+        MVPC   22(8,15),*+2(1),C' '    DEFAULT JOBNAME TO SPACES
+        MVI    30(15),C'A'             SET JOB CLASS
+*                                      (STATUS OPTION DEFAULTED TO 'RUN')
+        ST     R3,32(,15)              SET CPU TIME LIMIT
+        MVI    37(15),X'80'            FLAG CPU TIME LIMIT SET
+        OI     37(15),X'20'            SET CPU LIMIT EXPIRE OPTION:
+*                                                    X'40' - CANCEL
+*                                                    X'20' - PAUSE
+*                                                    X'00' - WARN
+        OI     37(15),X'04'            SET JOB DISPOSITION TO 'REQUEUE'
+*                                      (DUMP OPTION DEFAULTED TO "ON
+*                                      PROGRAM REQUEST ONLY")
+        PUSHA  0,0(,15)                POINT TO "PLIST" WITH STACK TOP
+*                                      WORD
+        MVI    0(15),1                 FLAG REQUEST TYPE: 1 = JOB
+*                                                         2 = PRINT
+        SVC    46   (SUBMIT)           ISSUE SVC
```

```
LAB       SUBMIT JOB,PLIST=MYPLIST,LIBRARY=PROCLIB,JOBNAME=MYJOB,        X
                 JOBCLASS=(R5),CPULIMIT=(,'CANCEL'),DUMP=DUMPOPT
+LAB      PUSHA  0,MYPLIST                POINT TO "PLIST" WITH STACK TOP
+*                                        WORD
+         MV     0(15),1                  FLAG REQUEST TYPE: 1 - JOB
+*                                                          2 - PRINT
+         MVC    MYPLIST+8(8),PROCLIB     SET LIBRARY NAME
+         MVC    MYPLIST+22(8),MYJOB      SET JOB NAME
+         MVC    MYPLIST+30(1),0(R5)      SET JOB CLASS
+         OI     MYPLIST+37,X'40'         SET CPU LIMIT EXPIRE OPTION:
+*                                                       X'40' - CANCEL
+*                                                       X'20' - PAUSE
+*                                                       X'00' - WARN
+         MVC    MYPLIST+31(1),DUMPOPT    SET DUMP OPTION
+         SVC    46      (SUBMIT)         ISSUE SVC

LAB       SUBMIT PRINT,FILENAME='MYFILE",LIBRARY=PRINTLIB,VOLUME=(R5),   X
                 PRTCLASS=(R2),FORM#=27,DISP='SAVE'
+LAB      PUSHN  0,44                     GET SPACE ON STACK FOR "PLIST"...
+         XC     0(44,15),0(15)           ... AND CLEAR IT TO ZEROES
+         MVC    0(8,15),*+10             SET FILE NAME
+         B      *+12                     BRANCH AROUND LITERAL
+         DC     CL8'MYFILE'              FILE NAME
+         MVC    8(8,15),PRINTLIB         SET LIBRARY NAME
+         MVC    16(6,15),0(R5)           SET VOLUME NAME
+         MVC    22(1,15),0(R2)           SET PRINT CLASS
+         MVI    23(15),27                SET FORM NUMBER
+         MVI    25(15),1                 DEFAULT NUMBER OF COPIES TO 1
+*                                        (HIGH ORDER BYTE ALREADY CLEARED)
+*                                        (STATUS OPTION DEFAULTED TO
+*                                        'SPOOL')
+         MVI    27(15),X'20'             SET DISPOSITION: X'40' - REQUEUE
+*                                                        X'20' - SAVE
+         PUSHA  0,0(,15)                 POINT TO "PLIST" WITH STACK TOP
+*                                        WORD
+         MVI    0(15),2                  FLAG REQUEST TYPE: 1 - JOB
+*                                                          2 - PRINT
+         SVC    46     (SUBMIT)          ISSUE SVC
```

Set Telecommunications Stream Options

Syntax

```
[label]  TCOPTION UFB={(register)}[,STREAM={READER }]
                      {expression}[        {PUNCH  }]
                                  [        {PRINTER}]

                  [,DEVTYPE={2780  }]
                  [         {3780  }]
                  [         {TCDIAG}]
                  [,COMP={YES}][,PRINT={NO }][,BLOCKED={YES}]
                  [      {NO }][       {YES}][          {NO }]
                  [,RECSIZE=integer]
                  [,TRANSMISSION=([TRANSPARENT,    ]      ]
                  [               [NONTRANSPARENT,]      ]

                  [               [BLOCKED,  ]           ]
                  [               [UNBLOCKED,]           ]

                  [               [UNPADDED,]            ]
                  [               [PADDED,  ]            ]

                  [               [COMPRESSED,  ]        ]
                  [               [UNCOMPRESSED,]        ]

                  [               [EBCDIC]               ]
                  [               [ASCII ])              ]
```

Restriction

None.

Function

Sets the TC stream options in the UFB. The TC stream options consist of 3 bytes, the data option, the transmit/receive option, and maximum record size. They are stored in the UFBTCDATAOPT, UFBTCXMITOPT, and UFBTCMAXRECSZ. The stream options are defined as follows:

TC data option:

bit 0	=	1	print format VS records in use
1	=	1	compressed VS record format
2	=	1	blocked VS record format
3-5			(reserved)
6-7	=	0	for card reader stream
	=	1	for card punch stream
	=	2	for printer stream
	=	3	(reserved)

TC transmit/receive option:

bit 0	=	1	perform code translation from EBCDIC to ASCII
1	=	1	compress transmitted record data
2	=	1	pad transmitted records to exact length with space codes
3	=	1	blocked transmitted records
4	=	1	transmit in transparent mode
5-7			(reserved)

The third byte in the TC stream option is equal to the maximum or exact transmitted record length minus one.

Operand Descriptions

UFB = The address of a User File Block (UFB), which may be supplied as a register specified in parentheses, where the register contains the UFB address, or as an expression not in parentheses, where the word addressed is assumed to begin the UFB.

STREAM = To identify the stream of the TC line, valid values are READER for cardreader, PUNCH for cardpuncher, or PRINTER for printer.

DEVTYPE = To identify the device type of the TC line, valid values are 2780, 3780 for IBM-2780, IBM-3780 batch TC stream, and TCDIAG for TC diagnostic use. This option does not take effect until the addressed UFB has been OPENed again (unlike the other options of TCOPTION, which are effective on the next DMS Function request).

PRINT = If YES, the corresponding bit in the data option will be set to 1; otherwise, to 0.

BLOCKED = If NO, the corresponding bit in the data option will be set to 0; otherwise, to 1.

COMP = If NO, the corresponding bit in the data option will be set to 0; otherwise, to 1.

TRANSMISSION = The bits in the transmit/receive option will be set according to the operand specified. For example, if TRANSPARENT is specified, the corresponding bit 4 in the transmit/received option will be set to 1; if NONTRANSPARENT is specified, the bit 4 in the transmit/received option will be set to 0.

RECSIZE = The 3rd byte in the TC stream option will be set to the integer value minus 1.

Example

```
LAB1  TCOPTION  UFB=(R2),STREAM=PUNCH,BLOCKED=NO,RECSIZE=10,      X
              TRANSMISSION=(NONTRANSPARENT,PADDED)
+LAB1 LR      1,R2                SET REGISTER 1
+     MVI     85(1),65            SET TC DATA OPTIONS
+     MVI     86(1),B'01111000'   SET TC XMIT OPTIONS
+     MVI     87(1),10-1          SET TC MAXIMUM RECORD SIZE
```

Get Date and Time

Syntax

```
[label]  TIME  [ JUL ][   HMS ]
               [ YMD ][ , CLK ]
```

Function

The current date is returned in the higher-addressed word (stack pointer plus four) of a two-word area pushed onto the stack, in one of the following forms:

JUL - as a packed number 00YYDDDF, where YY is the year, DDD the day in year, and F a hexadecimal 'F' (positive sign);

YMD - as a packed number 0YYMMDDF, where YY is the year, MM the month, DD the day, and F a hexadecimal 'F'.

If HMS is specified (or by default), the current time is returned in the lower-addressed word of the two-word area pushed onto the stack, in packed digits: HHMMSSth, where

HH is hours in day,
MM is minutes in hour,
SS is seconds in minute,
t is tenths of second in second,
h is hundredths in tenth.

The minimum time value is 00000000. The maximum is 23595999.

If CLK is specified, the current clock value is returned in the lower-addressed word of the two-word area pushed onto the stack, in binary, in 1/100 second units from the previous midnight.

Operand Descriptions

See the above function description. Omitted operands default to 'JUL' and 'HMS'.

Example

```
 LAB1   TIME   JUL
+LAB1   PUSHA  0,0              'JUL' REQUEST
+       PUSHA  0,0              'HMS' REQUEST
+       SVC    2 (TIME)
```

Generate a User File Block

Syntax

```
[label] UFBGEN  [PRNAME=p1]          [,DEVCLASS={DISK }]
                                                {PRT  }
                                                {WS   }
                                                {MTAPE}

[,MODE={OUT    }]      [,FORG={CONSEC }]    [,VLEN={YES}
       {IN     }              {INDEXED}           {NO }
       {IO     }              {ANY    }
       {EXTEND}
       {SHARED}

[,COMP={YES}]          [,PRINT={YES}]       [,PROG={YES}]
       {NO }                   {NO }               {NO }

[,NRECS=p2 ]           [,RECSIZE={integer}] [,BLKSIZE={integer}]
                                {0      }           {0      }
                                {ANY    }

[,BUFSIZE={integer}]   [,FORM={0     }]     [,PRTCLASS={A  }]
          {2048   }            {1-255}                {B-Z}

[,DEVNO=integer]       [,KPOS=p4 ]          [,KSIZE=p5 ]

[,DPACK={100  }]       [,IPACK={100  }]     [,NOVTOC={YES}]
        {1-100}                {1-100}              {NO }

[,VERIFY={YES}]        [,RELEASE={YES}]     [,FILENAME=p6 ]
         {NO }                  {NO }

[,LIBRARY=p7 ]         [,VOLSER=p8 ]        [,FILECLAS={0     }]
                                                   {#,A-Z}

[,NODISPLAY={YES}]     [,RECAREA=p9 ]       [,KEYAREA=p10 ]
              {NO }

[,ERRAD=p11 ]          [,EODAD=p12 ]        [,POOL={YES},BCT=p14]
                                                   {NO }

[,ALTCNT={0    } ,ALTAREA=p15]              [,{PAM}={YES}]
         {1-16}                              {BAM} {NO }

[,BLKAL={YES},NBLKS=p13]                    [,STREAM={READER }]
        {NO }                                        {PUNCH  }
                                                     {PRINTER}
```

```
[,TRANSMISSION=[{TRANSPARENT    }]  [,{BLOCKED   }]
                {NONTRANSPARENT}      {UNBLOCKED}

               [,{UNPADDED}]  [,{COMPRESSED   }]  [,{EBCDIC}]]
                 {PADDED  }     {UNCOMPRESSED}      {ASCII }

[,MCTYPE={2780  }]       [,EOD=EOV]
         {3780  }
         {TCDIAG}        [,LABEL= {[NL][,AL][,IL][,ANY]}]

[,DEN={556 }]            [,FSEQ={integer}]     [,VSEQ={integer}]
      {800 }                    {1      }             {1      }
      {1600}

[,ALLOWTAPE={YES}]       [,TRACK7={YES}]       [,HEADER={PARTIAL}]
             {NO }                    {NO }                 {FULL   }

[,ALLOWNL={YES}]         [,PARITY={EVEN}]      [,PLOG={YES}]
           {NO }                  {ODD }              {NO }
```

Function

Generates a User File Block (UFB) with the specified fields initialized. This macroinstruction does not produce executable code.

Operand Usage

The following operands are only used for Physical or Block Access Method:

PAM=YES
BAM=YES
BLKAL=YES
NBLKS=p13

The other operands are for the normally used Record Access Method. The following table summarizes their use.

Operand Usage Table (Record Access Method)

Legend:

R 1 = Required for Open processing. Can optionally be set by the program prior to OPEN.

0 1 = Optional for Open processing. Can also be set by the program before OPEN.

R 2 = Required for DMS functions. Can be set by the program before use.

0 2 = Optional for DMS functions. Can be set by the program before use.

Underlines are used to identify default values.

OPERAND USAGE TABLE (RECORD ACCESS METHOD)

Commonly used operands

	New Consecutive Disk Files value	New Indexed Disk Files value	Existing Disk Files value	New Work or Temp Files value	New Print Files value	Workstation Files values
PRNAME	R 1 1-8 char alphanumeric	R 1 1-8 char alphanumeric	R 1 1-8 char alphanumeric	R 1 1-8 char alphanumeric	R 1 1-8 char alphanumeric	R 1 1-8 char alphanumeric
DEVCLASS	R 1 DISK	R 1 DISK	R 1 DISK	R 1 DISK	R 1 PRT	R 1 WS
MODE	R 1 OUT	R 1 OUT	R 1 {IN,IO, } {EXTEND,} {SHARED }	R 1 OUT	R 1 OUT	R 1 IO
FORG	R 1 CONSEC	R 1 INDEXED	R 1 {CONSEC, } {INDEXED,} {ANY }	R 1 CONSEC [7]	R 1 CONSEC	R 1 CONSEC
VLEN	0 1 YES	0 1 YES	0 1 YES		R 1 YES	
COMP	0 1 YES	0 1 YES			R 1 YES	
PRINT	0 1 YES		0 1 YES		R 1 YES	
PROG	0 1 YES		0 1 YES			
NRECS [2]	R 1 numeric	R 1 numeric		R 1 numeric	R 1 (1,000,numeric)	
RECSIZE	R 1 numeric	R 1 numeric	R 1 {ANY,numeric}	R numeric	R {134,numeric}	R {1924, numeric}
BUFSIZE	0 1 n*2k	0 1 n*2k	0 1 n*2k	0 1 n*2k	0 1 n*2k	

Operands used for print files only

	New Consecutive Disk Files		New Indexed Disk Files		Existing Disk Files		New Work or Temp Files		New Print Files		Workstation Files
	value		value		value		value		value		value
FORM	5								0 1	{0,1-255}	
PRTCLASS	5								0 1	{A,B-Z}	
DEVNO									0 1	numeric	
Operands used for new indexed files only											
KPOS			R	numeric							
KPOS			R	numeric							
KSIZE			R 1	numeric							
DPACK			0 1	numeric 1-100							
IPACK			0 1	numeric 1-100							
Special purpose operands											
NOVTOC	3	0 1 YES				0 1 YES					
VERIFY	4	0 1 YES		0 1 YES				0 1 YES		0 1 YES	

Operands used to default run-time assignments

		New Consecutive Disk Files		New Indexed Disk Files		Existing Disk Files		New Work or Temp Files		New Print Files		Workstation Files
		value		value		value		value		value		value
5	0	1-8 char	0	1-8 char	0	1-8 char	R	#AAAA				
FILENAME	1	alphanumeric	1	alphanumeric	1	alphanumeric	1	or ##AAAA where AAAA is 1-4 char alphanumeric				
		(See Note 5)						See NOTE (1)				
5	0	1-8 char	0	1-8 char	0	1-8 char						
LIBRARY	1	alphanumeric	1	alphanumeric	1	alphanumeric						
5	0	1-6 char	0	1-6 char	0	1-6 char						
VOLSER	1	alphanumeric	1	alphanumeric	1	alphanumeric						
5	0	{#A-Z}	1	{#,A-Z}			R	{#,A-Z}	R	{#,A-Z}		
FILECLAS	1		1				1		1			
6	0	YES	0	YES	0	YES	R	YES	R	YES	R	YES
NODISPLAY	1		1		1		1		1		1	

Operands used for DMS requests

	New Consecutive Disk Files		New Indexed Disk Files		Existing Disk Files		New Work or Temp Files		New Print Files		Workstation Files	
		value		value		value		value		value		value
RECAREA	R 2	address in segment 2	R 2	address in segment 2	0 2	address in segment 2	R 2	address in segment 2	R 2	address in segment 2	R 2	address in segment 2
KEYAREA			R 2	address in segment 2	0 2	address in segment 2						
ERRAD	0 2	instruction address	0 2	instruction address	0 2	instruction address	0 2	instruction address	0 2	instruction address	0 2	instruction address
EODAD	0 2	instruction address	0 2	instruction address	0 2	instruction address	0 2	instruction address	0 2	instruction address	0 2	instruction address

Notes:

(1) The escape characters # and ## are used to request unique name generation and to identify work files (#) and temporary files (##) to the system. Work Files and temporary files are placed in the user's workfile library regardless of what is supplied for library and volume. The work files are automatically scratched when the file is closed. The temporary files are automatically scratched at the end of the run.

(2) NRECS should preferably be set by the program as a value determined after opening the associated input file(s) and learning its (their) size.

(3) See the description of NOVTOC files. When NOVTOC is used, FORG=CONSEC, VLEN=NO, COMP=NO, and FILENAME, LIBRARY, and FILECLAS are ignored.

(4) The use of the VERIFY option significantly degrades performance. Its use, unless specifically intended, is not recommended.

(5) These operands represent run-time parameters ultimately determined via GETPARM. Unless required to identify WORK or TEMP files, these operands serve only to provide default values. If left unspecified, defaults are provided by OPEN using values supplied via the SET command or via system conventions.

(6) Causes a GETPARM type "ID" to be issued, thus suppressing user interaction. Should be used to minimize transactions for fixed file specifications only.

(7) Other file organizations including INDEXED, VLEN, COMP, PRINT, and PROG, are supported but apparently not very useful. If these organizations are used, other supplied operands must be consistent.

Notes on operands

PRNAME= The parameter reference name is the fundamental identifier used to locate or solicit run-time parameter information. The prnames used should be indicative of function.

FORG= The file organization parameter is used for existing files to verify file organization. If FORG=ANY is specified, any file organization is accepted.

FORG=ANY can be specified for tape in INPUT mode. For unlabelled tape, FORG is set to consecutive.

VLEN=
PRINT=
PROG=
These attributes are used for existing files to limit acceptance to files of the indicated attribute.

NRECS=
The number of records is defaulted to 1000 for existing files.

RECSIZE=
BLKSIZE=
The record size (or maximum record size) and block size are used for existing files to verify this file attribute. A RECSIZE (BLKSIZE) of zero is used to accept any record (block) size.

BUFSIZE=
The buffer size option is used to increase efficiency for sequential processing. See Chapter 7, DMS, for details.

DEVNO=
The device number option can be used for print files to request printing on a specific printer.

IPACK=
DPACK=
These options are used to specify the relative percentage of data to space (packing) desired on a new indexed file. If not specified, system default values are used.

PAM=
Optional request for Physical Access Method support or multiple-line printing. Defaults to PAM=NO.

BAM=
Optional request to process a disk file as if it had 2048-byte logical records, irrespective of the record size recorded in its file descriptor record. Defaults to BAM=NO.

NOVTOC=
Optional file attribute for diskette only. Specify only for unstructured diskette.

RELEASE=
If set to YES in output mode, the unused disk space allocated for the file will be released at file close.

RECAREA=
KEYAREA=
These operands must be acceptable in "DC A(pn) assembler statements. They may be written only when the UFB is generated in a segment 2 'static' area.

DEVCLASS=
Valid options are DISK, MTAPE, WS, TC, or PRT.

VERIFY=
Requests read-after-write verification on a disk file.

BLKAL=
Allocate space for a new disk file in number of blocks, as specified in UFBNBLKS (see NBLKS=operand), rather than in number of logical data records.

POOL= Buffer pooling requested. The BCT operand addresses a Buffer Control Table in the user-modifiable segment, as created by the BCTGEN macroinstruction.

ALTCNT= An integer between 0 and 16. If not 0, ALTAREA operand must be supplied. This is the number of alternate indices processable for the file.

ALTAREA= Address of AXD1 block, as generated by the AXDGEN macroinstruction.

STREAM= Sets the TC STREAM DATA option in UFBTCDATAOPT.

TRANSMISSION= Sets the TC STREAM TRANSMIT/RECEIVE options in UFBTCXMITOPT (see also TCOPTION macro).

MCTYPE= Sets the microcode type for programmable devices. Currently, valid options are 2780 and 3780 for IBM-2780 and IBM-3780 batch telecommunications emulation, and TCDIAG for telecommunications diagnostic use.

EOD= Forces EOD exit when a data management operation reaches the end of a tape volume in INPUT mode and an EOV1 trailer label is detected.

LABEL= Magnetic tape label types allowed (NL for no-label, AL for ANSI-label, IL for IBM-Label; none, one, or more than one may be specified).

DEN= Magnetic tape density: 556 for 556 BPI tape, 800 for 800 BPI tape, and 1600 for 1600 BPI tape. The default value is 1600 BPI.

FSEQ= Tape file sequence number.

VSEQ= Tape volume sequence number for multiple-volume tape file.

ALLOWTAPE= If set to YES, OPEN will allow tape as an alternative device for disk file.

TRACK7= If YES is specified, then 7-track tape is indicated. The default is NO. The user must specify a non-zero value for tape density in the UFB in the case of a 7-track tape.

HEADER= This operand supports IBM DOS labelled tapes. If FULL is specified, then both HDR1 and HDR2 file labels are present on the tape. If PARTIAL is specified, then only HDR1 is present. If HEADER=FULL is specified, and no HDR2 is found on the tape, OPEN will cancel with an indication of an invalid label type. If HEADER=PARTIAL

is specified, but a HDR2 label is found on the tape, OPEN will proceed to open the file using structural information from the HDR2.

When only HDR1 is present, the user must provide valid information about file organization, record length, and block size in the UFB.

ALLOWNL= Allows non-labelled tape. The default is NO.

PARITY= If EVEN is specified, then the tape uses even parity. If ODD is specified, then the tape uses odd parity. EVEN is the default.

NODISPLAY= If YES is specified, then the OPEN SVC will not issue a GETPARM to the user's workstation for CANCEL messages or for respecification messages.

PLOG= If YES is specified, indicates that a file prologue will be present. Valid only for Word Processing files. This operand is applicable only when OPEN mode is OUTPUT, and will be ignored for any other OPEN mode. The default is to NO.

Write a Record

Syntax

```
[label]  WRITE  [EOM,]  UFB={(register)}  [,COND=number]
                [EOT,]     {expression}
```

Function

Writes the next sequential record to a consecutive or indexed disk, magnetic tape, or printer file. The file must be open in OUTPUT or EXTEND mode, or the specified record (Key addressed by UFBKEYAREA) to an indexed disk file open in IO or SHARED mode. For indexed disk files, open in OUTPUT or EXTEND mode, the key in the record to be written is checked to insure that it is greater than any key already in the file. If not, a record sequence error is indicated.

A possible invalid-key condition which can be indicated in file status bytes (UFBFS1, UFBFS2) is:

- 21 - Record sequence error (indexed files only)
- 22 - Duplicate Key (indexed files only)
- 24 - Boundary violation (primary extent size exceeded in output mode - indexed files only)

Possible error conditions which can be indicated in file status bytes are:

- 30 - Permanent I/O error
- 34 - Boundary violation (consecutive files in output or extend mode; indexed files in I/O or shared mode)
- 95 - Invalid function or function sequence
- 96 - Invalid data area location or alignment
- 97 - Invalid length for device

An invalid-key condition results in return to the address in UFBEODAD, with the normal return address in register 0. Other exceptional and error conditions result in return to the address in UFBERRAD, with the normal return point address in register 0. If UFBEODAD is zero, UFBERRAD is used in its place. If UFBERRAD is zero as well, any exceptional condition results in abnormal termination of the program.

Operand Descriptions

EOM= This option indicates that the data transmitted by the WRITE function is to be followed with a telecommunications end-of-message character. (Pertains only to batch telecommunications devices.)

EOT= This option indicates that a telecommunications end-of-transmission signal is to be transmitted, following any data specified. (Pertains only to batch telecommunications devices.)

UFB= The address of a User File Block (UFB), which may be presented as a register specification in parentheses, where the register contains the UFB address, or as an expression not in parentheses, where the word at the address designated by the expression is assumed to begin the UFB.

COND= If specified, the number or absolute expression becomes the first operand of the JSCI instruction by which the WRITE function is entered. Thus the WRITE is made conditional. The default is COND=15. Register 1 is loaded with the UFB address even when the condition is not satisfied.

Execute Physical I/O

Syntax

```
[label]   XIO [     OFB= {address   }]     [,COMMAND= {address   }]
              [          {(register)}]     [          {(register)}]

              [   ,MEMA= {address   }]     [ ,BLKNUM= {address   }]
              [          {(register)}]     [          {(register)}]

              [,BLKSIZE= {address   }]     [  ,PLIST= {address   }]
              [          {(register)}]     [          {(register)}]

              [,RELEASE]

              [  ,VOLIO= {YES}          ,VCB= {address   }]
              [          {NO }                {(register)}]

              [,MLPRINT= {YES}         ,FORM= {LIST}]
              [          {NO }                {EXEC}]

              [,UCPRINT= {YES}] [,DEVSTATUS= {CLEAR   }]
              [          {NO }] [            {CHECK   }]
                                [            {NOCHECK}]
```

Restrictions

XIO is intended for use by Data Management System routines. XIO with the VOLIO option is allowed only when requested from within System Mutual Exclusion (SME) state or when addressed to a disk volume placed in initialization state by the issuing task. It is valid only for disk operations.

Function

Normal (without VOLIO option):

Validates disk extents; acquires available physical pages of memory for input operations if the virtual pages referenced are not in main memory; "short-term fixes" the virtual data page or pages in physical pages during the I/O operation; constructs indirect data address lists for workstation and disk operations; insures that the "change" bit in the Page Frame Table for each modified page is set when read-type I/O is accomplished; enters the System Start I/O Routine to initiate the operation; returns to the issuer on completion of its functions with a one-word return code field replacing the input parameters on the top of the stack, as described below.

With VOLIO option:

Validates Volume Control Block address, disk block numbers, and data address; validates that usage of the VOLIO option is to be allowed; translates memory address; converts block on volume to disk address; constructs IOCW (from COMMAND operand, converted MEMA operand, converted BLKNUM operand) in the IORE contained in this VCB; "fixes" data page if required, as described above; sets "change" bit if required; enters System Start I/O Routine to initiate the operation; returns to issuer with return code field on the top of the stack.

Low-order halfword of return code field - binary return codes:

0 - Success
4 - Truncation at end-of-extent (non-VOLIO disk only)
8 - Truncation at end-of-cylinder or end-of-track (disk only)
12 - Starting block number beyond end-of-file (non-VOLIO disk) or beyond end-of-volume (VOLIO disk)
16 - Invalid data address or data length (Data address for disk must be page-aligned; for other devices word-aligned. Virtual memory area encompassed by data address through data address plus block size minus one must either be in the segment 2 I/O buffer area or entirely above the XIO parameter list on the stack if the XIO is issued from unprivileged state)
20 - Second XIO on file without intervening CHECK
24 - TC XIO attempted on an OFB that was not created as the result of an 'IPOPEN' on an IPCB
28 - TC XIO attempted on a device reserved exclusively by another task
32 - XIO has been issued to an inoperative workstation and the I/O has not been issued (Bit 5 of option flag must be set for issuance of this return code)
36 - TC XIO attempted on a peripheral processor (DLP) reserved exclusively by another task
40 - Write XIO attempted to file opened in "WPSHARE" mode, file not locked.
44 - Read XIO attempted to file opened in "WPSHARE" mode, file locked by another user.

High-order halfword of return code field - residual block counts:

Return codes 4, 8 - block size specified minus number of bytes actually read or written. All other return codes are always zero.

Note: If return codes 0, 4, or 8 are set, the I/O operation is queued for initiation and a CHECK must be issued to test for completion. If return codes 12, 16, or 20 are set, the operation has been suppressed. XIO never waits for I/O completion.

Operand Descriptions

OFB= The address of the Open File Block (OFB) for file involved in the I/O operation. The OFB is supplied when the file is OPENed. This operand is not used in conjunction with the VOLIO option.

COMMAND= The address of the value to be placed in the Command Byte of the I/O Command Word (IOCW) constructed by the XIO SVC. The command byte specifies the operation to be performed. Possible values are contained in descriptions of the IOCWs for the various commands.

MEMA= The address of a 4-byte area containing a virtual data address for the I/O operation in its low-order three bytes, to be translated to a physical address and then placed in the IOCW, or as a register specification in parentheses where the register contains the virtual address.

BLKNUM= For disk I/O, the address of a three-byte area containing the block number (from zero) within the file of the block to be read from the file. If the 'VOLIO' option is specified, or if an unstructured diskette devcie is being referenced, this is to be the block on volume, from block 0.

BLKSIZE= For all read or write operations, the address of a halfword area containing the length in bytes for the operation (or maximum length, as for magnetic tape)

PLIST= The address of a 16-byte area containing the parameter list for XIO. If this operand is supplied, any other operands are used to modify the parameter list after it has been moved to the stack. The original copy is not modified.

RELEASE Specified on a disk or tape write operation when it is desired to make the fixed page frames available after the operation without preserving their contents (i.e., without pageout).

VOLIO= If YES is specified, then perform volume-oriented disk I/O without extent limitations, as described above. Valid only for disk files, and only when requested by system routines in System Mutual Exclusion (SME) state or when the accessed volume is mounted for initialization by the issuing task.

VCB= Address of Volume Control Block for a disk volume. Required with VOLIO option unless PLIST= is supplied or the FORM=EXEC option is specified. Allowed only with VOLIO option. Register 1 will be modified if this operand is written as an expression not in parentheses.

MLPRINT= If YES is specified, then requests a block-print opeation of one or more lines. Record-length bytes must be provided in the data area if this option is not specified. Ignored if the operation is not directed to a printer. Data must be 2K-aligned and is not moved to the device's resident print buffer.

FORM= If EXEC is specified, the parameter list is assumed to already be enstacked. The supervisor call is generated. If other operands are supplied, they are used to modify the existing parameter list. The

VOLIO=YES and RELEASE operands must be specified if required, even if the parameter list already contained these options.

If LIST is specified, the parameter list is created on the stack, but the supervisor call is not generated. The RELEASE operand is normally not useful on an XIO macroinstruction with FORM=LIST.

UCPRINT= If YES is specified, then upper case printing is used. The default is NO.

DEVSTATUS= This operand is intended for the use of hardware diagnostics personnel when simulating error conditions on serial workstations and printers.

If CLEAR is specified, XIO will reset two fields in the Unit Control Block (UCB), i.e., UCBSTATNOTOP and UCBSTATNOCODE, thus permitting I/O to a device which is being simulated to malfunction.

If CHECK is specified and an XIO is issued to an inoperative workstation, then a return code of 32 is generated and the I/O is not issued.

If NOCHECK is specified, then any attempts at I/O to a malfunctioning workstation will cause the task to wait for the device to become operational. NOCHECK is the default value.

Examples

```
(1)   LAB1  XIO    COMMAND=RDCMD,PLIST=XIOPARM
      +LAB1 PUSHC  0(16,0),XIOPARM
      +     MVC    5(1,SP),RDCMD
      +     SVC    3 (XIO)

(2)   LAB2  XIO    OFB=R(1),MEMA=(R2),BLKNUM=UFBBUFBLOCK,                X
            BLKSIZE=UFBBLKSIZE,COMMAND=WRCMD,RELEASE
      +LAB2 PUSHN  0,16
      +     MVC    8(2,15),UFBBLKSIZE
      +     MVC    12(3,15),UFBBUFBLOCK
      +     ST     R2,4(,15)              MEMA
      +     ST     R1,0(,15)              OFB
      +     MVC    4(1,15),WRCDM          COMMAND
      +     MVI    0(15),X'80'            RELEASE
      +     SVC    3 (XIO)

(3)   LAB3  XIO    COMMAND=RDCMD,PLIST=XIOPARM,VOLIO=YES
      +LAB3 PUSHC  0(16,0),XIOPARM
      +            MVC    5(1,SP),RDCMD
      +            OI     0(SP),X'40'     VOLIO
      +            SVC    3 (XIO)
```

Transmit Intertask Message

Syntax

```
[label] XMIT MESSAGE = {(register)}
                       {expression}

                ,PORT= {(register)}
                       {expression}
                       {'string'  }

                [,NOWAIT] [,OTHERTASK]
```

Function

Queues the message at the specified address for receipt by the owner of the specified message port. The CHECK macroinstruction is used to accept receipt of a message.

Return codes are placed in the word on the stack top as follows:

- 0 - Successful
- 4 - No receiving message port with the specified name
- 8 - Unable to insert message in receiving port's message buffer ('NOWAIT' option only)
- 12 - Unable to insert message in receiving port's message buffer due to receiving port's use of PRIVILEGED option
- 16 - Message not transmitted; OTHERTASK option was specified and the designated message port belongs to the XMIT-issuing task

Operand Descriptions

MESSAGE= The address of a message, which may be anywhere in the issuer's address space. The first two bytes of the message area must contain the length of the message in binary, including these bytes, and may not be greater than 2016.

May be specified as a register in parentheses containing the address of the message, or as an expression addressing the message.

PORT= The four-character name of the receiving message port, which may be specified as an address expression, or as a register designation where the register contains the address of the four characters in memory, or as a literal value in quotes.

NOWAIT If specified, return to issuer immediately with return code 8 if there is insufficient space in the receiving port's message buffer to insert the message.

OTHERTASK If specified, return to issuer immediately with return code 16 if the designated receiving message port belongs to XMIT-issuing task.

Example

```
 LAB1   XMIT   PORT='DBMS',MESSAGE=(R2)
+LAB1   PUSHC  0(4,0),*+10
+       B      *+8
+       DC     C'DBMS'
+       PUSH   0,R2
+       MVI    0(15),B'00000000'
+       SVC    36 (XMIT)
```

CHAPTER 5: CONTROL BLOCKS

5.1 INTRODUCTION

This chapter documents the internal control blocks of the VS Operating System which are of interest for the general user. The following control blocks are described, in Assembler language format, and with offset locations:

AXD1	IORE
BCE	OFB
BCTBL	TPLAB
EXTRD	TPLB2
FDR1	UFB
FDR2	VOL1

Blocks FDR1 and FDR2 constitute part of the disk Volume Table of Contents (VTOC), and normally are present on disk only. Blocks TPLAB and TPLB2 are tape labels. Block VOL1 is the tape and disk volume label. All other blocks are kept in the user's modifiable segment (Segment 2), when present.

The control blocks change frequently. They can be assembled by the user with the following lines of code:

```
<ctl block>
END
```

where <ctl block> is replaced by a block name.

A second set of control blocks, intended for operating system use only, is kept in the protected system memory segment (Segment 0). They consist of the following blocks: CMSG, DBTB, DPT, ETCB, FLUB, FMSG, LCB, MCB, PFB, PFSA, PFT, PPB, PRB, PT, PXE, RMSG, STMB, SVCT, TCB, TQEL, UCB, and VCB. These control blocks are described in a separate document.

```
                AXD1
000000 AXD1          DSECT
       *
       *        THE ALTERNATE INDEX DESCRIPTOR BLOCK (AXD1) DESCRIBES THE
       *        ALTERNATE INDEX STRUCTURES OF AN INDEXED FILE. AN INDEXED
       *        FILE HAS AN AXD1 BLOCK IF AND ONLY IF FLAG FDR1FLAGSALTX
       *        IS SET IN ITS LABEL (FDR1). THE AXD1 BLOCK CONTAINS
       *        UP TO 16 (64) ALTERNATE INDEX DESCRIPTIONS (AXD1ENTRY). THE
       *        NUMBER OF DESCRIPTIONS IS CONTAINED IN FDR1ALTXCNT OF THE
       *        FDR1 RECORD.
       *
       *        THE AXD1 IS LOCATED IN BLOCK NUMBER ZERO OF THE FILE.
       *        THE AXD1 IS DIVIDED INTO 4 AREAS:
       *             1. BLOCK DESIGNATOR AREA (AXD1BL)
       *             2. DMS PROCESSING AREA (AXD1MASK TO AXD1ENTRY)
       *             3. AXD ENTRIES (ONE AXD ENTRY PER ALT-INDEX)
       *             4. SPARE AREA (UP TO END OF 2K BLOCK)
       *        AREAS 1-3 ARE HELD IN THE AXD1-AREA (POINTED TO BY UFBALTPTR)
       *        DURING FILE PROCESSING.
       *
       *        DATE 3/28/79
       *        VERSION 4.0
       *
       * BLOCK DESIGNATOR AREA:
000000 AXD1BEGIN                DS  0F
000000 AXD1BL                   DS  BL4  BLOCK TYPE DESIGNATION
       *                                 AXDBL1 MUST EQUAL XL4'2'
       *
       * DMS PROCESSING AREA:
000004 AXD1MASK                 DS  BL8  BITS ON INDICATE ALTERNATE
       *                                 INDEX STRUCTURES (NUMBERED
       *                                 1 TO 16) PRESENT
       *                                 (INITIAL IMPLEMENTATION OF
       *                                 2-BYTE MASK ONLY)
00000C AXD1UFB                  DS  A    POINTER TO UFB FOR THIS FILE
       *                                 AFTER THE FILE HAS BEEN OPENED
000010 AXD1ALTINX               DS  BL1  ORDINAL INDEX NUMBER FOR READ
000011 AXD1FLAGS                DS  BL1  DMS FLAG BYTE
       AXD1FLAGSOK              EQU X'80' ALTERNATE INDEX STRUCTURES HAVE
       *                                   BEEN CREATED WHEN FLAG SET
       * THE FOLLOWING FLAGS ARE USED FOR DMS PROCESSING (0 IN LABEL)
       AXD1FLAGSQ               EQU X'04' START QUALIFIED OPTION
       AXD1FLAGSTYPER           EQU X'02' TYPE R SAVEAREA IN USE
       AXD1FLAGSTYPEV           EQU X'01' TYPE V SAVEAREA IN USE
       **
000012 AXD1MSIZE                DS  BL1   SIZE OF MASK PER FILE
       *                                 VALUE FROM 2-8 BYTES (MUST BE 2
       *                                 FOR FIRST IMPLEMENTATION)
000013 AXD1DUPINX               DS  BL1  ORDINAL INDEX NUMBER OF THE
       *                                 ALT-TREE HAVING DUPLICATED KEY
```

```
       * MINIMUM AXD1-AREA FOR SHARED MODE ENDS HERE.
       * AXD1MASK, AXD1MSIZE, AND AXD1ALTINX ARE REQUIRED.
       *
000014 AXD1BCB                  DS  BL16 BCB FOR DMS PROCESSING (SEE UFB)
000024 AXD1PMASK                DS  BL8  MASK OF VALID ALTERNATE ACCESS
       *                                 PATHS (SET AT FILE CREATION ONLY)
       *
       * THE FOLLOWING FIELDS ARE INTERMEDIATE OUTPUT MODE FIELDS
       *
00002C AXD1ORECSIZE             DS  H    WORK RECORD - MAX LENGTH
00002E AXD1OFLAGS               DS  BL1  OUTPUT FLAGS (RESERVED)
00002F AXD1OSTART               DS  BL3  FIRST BLOCK CONTAINING WORK RECORDS
000032 AXD1ONRECS               DS  BL3  TOTAL COUNT OF WORK RECORDS
000035 AXD1OEBLK                DS  BL3  LAST USED BLOCK NUMBER IN PRIMARY
       *                                   TREE (ALT-TREE TO AXD1EBLK+1)
000038                          DS  H    USED FOR AXD1ADMSMASK (SEE BELOW)
00003A AXD1OSPARE               DS  BL2  RESERVED IN OUTPUT MODE
       **
00003C              ORG AXD1ORECSIZE
       * THE FOLLOWING FIELDS ARE USED FOR DMS PROCESSING (EXISTING FILES)
       **
00002C AXD1SAVEADR              DS   A  SAVE AREA ADDRESS (TYPE V)
000030 AXD1SAVELTH              DS   H  SAVE AREA LENGTH (TYPE V)
000032              ORG AXD1ORECSIZE
       * THE FOLLOWING 3 FIELDS ARE USED FOR SAVE AREA TYPE S
00002C AXD1SKEYSIZE             DS   BL1  SAVED PRIMARY KEYSIZE
00002D AXD1SHXBLK               DS   BL3  SAVED PRIMARY ROOT BLOCK NUMBER
000030 AXD1SEREC                DS   H    SAVED PRIMARY LEVEL COUNT
       *
000032 AXD1ENTOFF               DS   H    OFFSET OF ACTIVE AXD1ENTRY(IN AXD1)
000034 AXD1PTRN                 DS   BL3  NEXT SEQUENTIAL BLOCK (ALT-TREE)
000037 AXD1CURINX               DS   BL1  ORDINAL NUMBER ASSOCIATED WITH
       *                                  BLOCK IN AXD1BCB
000038 AXD1ADMSMASK             DS   H    ALTERNATE INDEX PATH MASK WITHIN
       *                                  THE VIEW (ADMS USE ONLY)
00003A AXD1EXSPARE              DS   BL2  SPARE - ALL FILES
       **
       *
       ********************************************************************
       * AXD1MASK AND AXD1ALTINX ARE THE ONLY FIELDS IN THE AXD1-AREA WHICH
       * MAY BE MODIFIED BY THE USER-PROGRAM WHILE THE FILE IS OPEN.
       *
       * FOR EXISTING FILES, NO FIELDS IN THE AXD1-AREA ARE USER-SUPPLIED
       * PRIOR TO ISSUING SVC OPEN.
       *
       * FOR OUTPUT MODE, USER-PROGRAM FILLS IN THE REQUIRED AXD1-AREA WITH:
       *        AXD1MSIZE (THE ACCESS MASK PREFIX SIZE);
       *        AXD1KEYPOS, AXD1KEYSIZE, AXD1EFLAGS, AND AXD1XORD
       *              FOR EACH AXD1ENTRY (COUNT IN UFBALTCNT).
       ********************************************************************
```

```
       *
       * AXD ENTRIES:
00003C AXD1ENTRY                DS  0XL28  UP TO 64 ENTRIES
       *                                   (EACH A DESCRIPTION OF ONE
       *                                   ALTERNATE INDEX STRUCTURE;
       *                                   UNUSED ENTRIES ZERO-FILLED)
00003C AXD1XORD                 DS  HL1    ORDINAL NUMBER  (STARTING FROM 1)
       *                                   IDENTIFYING THIS INDEX STRUCTURE
       *                                   (CORRESPONDS TO BIT IN
       *                                   AXD1MASK)
00003D AXD1EFLAGS               DS  BL1    OPTION FLAGS
       AXD1EFLAGSDUPS           EQU X'80'  DUPLICATE KEYS ALLOWED
       AXD1EFLAGSKCOM           EQU X'40'  KEY COMPRESSION IN INDEX
       *                                    (NOT IN FIRST VERSION)
       * THE FOLLOWING FLAGS ARE USED FOR DMS PROCESSING (0 IN LABEL)
       AXD1EFLAGSACT            EQU X'02'  INDICATES THIS ALT-TREE IS THE
       *                                    ACTIVE ALT-TREE DURING PROCESSING
       AXD1EFLAGSUP             EQU X'01'  INDICATES AXD1PTRD, AXD1XLEVELS
       *                                    OR AXD1HXBLK HAS BEEN MODIFIED
       *                                    DURING ALT-TREE PROCESSING
00003E AXD1XLEVELS              DS  H      NUMBER OF LEVELS OF THIS
       *                                   ALTERNATE INDEX STRUCTURE
       *                                   EXCLUDING LOWEST LEVEL
000040 AXD1KEYPOS               DS  H      KEY POSITION IN RECORD
000042 AXD1KEYSIZE              DS  HL1    KEY LENGTH
000043 AXD1HXBLK                DS  FL3    BLOCK-IN-FILE OF ROOT BLOCK
       *                                   OF THIS ALTERNATE INDEX
000046 AXD1NRECS                DS  BL3    ITEM COUNT - LOW LEVEL OF TREE
000049 AXD1PTRD                 DS  FL3    FIRST BLOCK OF LOW LEVEL
       *                                   OF THIS ALTERNATE INDEX
       *                                   (ALTERNATE KEY SEQUENCE)
00004C AXD1ESPARE               DS  BL12   (RESERVED IN EACH ENTRY)
       AXD1ENTRYEND             EQU *
       AXD1ENTRYLENGTH          EQU AXD1ENTRYEND-AXD1ENTRY
       *
000058                     ORG AXD1ENTRY+64*L'AXD1ENTRY
00073C AXD1SPARE3               DS  XL196  (RESERVED)
       *
       AXD1END                  EQU *
       AXD1LENGTH               EQU AXD1END-AXD1BEGIN
```

```
                BCE
000000 BCE          DSECT
       *
       *        THE BUFFER CONTROL ENTRIES (BCE) ARE CONTAINED IN THE BUFFER
       *        CONTROL TABLE (BCTBL). THERE IS ONE BCE PER 2K BUFFER IN A
       *        DATA MANAGEMENT BUFFER POOL. BCTNBUF (WHICH AGREES WITH
       *        OFBBCOUNT FOR AN ACTIVE BUFFER POOL) INDICATES THE NUMBER
       *        OF BUFFER CONTROL ENTRIES PER BCTBL.
       *
       *        DATE 3/28/79
       *        VERSION 4.00
       *
000000 BCEBEGIN                  DS  0F          (FULLWORD ALIGNMENT)
000000 BCEOFB                    DS  A           OFB ADDRESS
000004 BCEBUFCMD                 DS  0BL1        COMMAND BYTE
000004 BCEBUFADR                 DS  A           BUFFER MEMORY ADDRESS
000008 BCEBUFDATAL               DS  H           IO-LENGTH (2K)
00000A BCESPARE                  DS  H           OFFSET (UNUSED IN BCE)
00000C BCEBUFBLOCK               DS  FL3         BLOCK WITHIN
       *                                         FILE OF BUFFERED DATA
00000F BCEBCBFLAGS               DS  BL1         FLAGS
       BCEBCBFLAGSLOD            EQU X'01'       BUFFER CONTENTS VALID
       BCEBCBFLAGSTOR            EQU X'02'       BUFFER TO BE REWRITTEN
       BCEBCBFLAGSIO             EQU X'04'       BUFFER I/O IN PROGRESS
       BCEBCBFLAGSREF            EQU X'80'       REFERENCE BIT
       *                                         BIT=1 ON ANY READ/WRITE
000010 BCEKEYHI                  DS  CL12        TRUNCATED HI KEY VALUE
       *                                         (TYPE D)
       * BLOCK TYPE (BCETYPE) CONTAINS INTERNAL AND EXTERNAL VALUES
       * DEPENDING ON FILE ORG (INDEXED FILES HAVE **** NO **** BLOCK TYPE
       * BYTE IN THE BLOCK; THUS I,D,A BELOW ARE INTERNAL TYPES.)
00001C BCETYPE                   DS  CL1         BLOCK TYPE (ASCII CHAR)
       * BLOCK TYPE VALUES (INTERNAL) FOR INDEXED FILES
       BCETYPEI                  EQU C'I'        INDEX BLOCK
       *                                         (CONTAINS INDEX ITEMS)
       BCETYPED                  EQU C'D'        DATA BLOCK
       *                                         (CONTAINS DATA RECORDS)
       *                                         BCEKEYHI/LOW SET IF TYPE = D
       BCETYPEA                  EQU C'A'        AVAILABLE BLOCK (CHANGED TO
       *                                         TYPE I OR D IF USED
       *                                         BY BLOCK SPLIT)
       BCETYPES                  EQU C'S'        BLOCK FROM LOW-LEVEL OF AN
       *                                         ALTERNATE TREE
00001D BCEWT                     DS  BL1         STARTING WEIGHT VALUE
00001E BCEAGEWT                  DS  BL1         AGED WEIGHT VALUE
00001F BCESPARE1                 DS  BL1         SPARE
000020 BCEIOCHN                  DS  A           CHAIN FOR BCE'S WITH I/O
       *                                         IN PROGRESS
000024 BCEKEYLOW                 DS  CL12        TRUNCATED LOW KEY VALUE
       *                                         (TYPE D)
000030 BCEEXPAND                 DS  BL8         BCE EXPANSION
       * EXPANSION = 12 (TRUNC KEYS =12), PLUS 4 (CHN BCE PER UFB)+4 EXTRA
       BCELENGTH          EQU *-BCEBEGIN        BCE LENGTH (=56)
```

```
                BCTBL
000000 BCTBL          DSECT
       *
       *        THE BUFFER CONTROL TABLE (BCTBL) IS ADDRESSED FROM THE USER
       *        FILE BLOCK (UFB), AND CONTAINS A HEADER DEFINING A DATA
       *        MANAGEMENT BUFFER POOL AND BUFFER CONTROL ENTRIES (BCE)
       *        DEFINING THE CONTENTS OF EACH BUFFER IN THE POOL.
       *
       *        DATE 3-28-79
       *        VERSION 4.00
       *
000000 BCTBLBEGIN                 DS  0F          (FULLWORD ALIGNMENT)
       *
       *** BUFFER CONTROL TABLE
       *
000000 BCTBLNBUF              DS   0HL1           COUNT OF BUFFERS (BCE'S)
000000 BCTBLHITCT             DS   A              HIT-COUNT (READ)
000004 BCTBLLOCK1             DS   A              BCE LOCK1 (DMS INTERNAL)
000008 BCTBLREPLNUM           DS   0HL1           CIRCULAR BCE NUMBER (SCAN)
       * BCTBLHITCT AND BCTBLMISSCT INDICATE PERCENTAGE OF READ OPERATIONS
       * HANDLED WITHIN THE BUFFER POOL (WITHOUT PHYSICAL IO OPERATION).
000008 BCTBLMISSCT            DS   A              MISS-COUNT (READ)
00000C BCTBLFILECT            DS   BL1            COUNT OF FILES USING BCT
00000D BCTBLFLAGS             DS   BL1            BCTBL FUNCTION FLAGS
       BCTBLFLAGSEXT          EQU  X'80'          INTERNAL FLAG FOR
       *                                          EXTRACT FUNCTION
       BCTBLFLAGSRPL          EQU  X'40'          GET REPLACEMENT BUFFER
       *                                          WITHOUT IO OPERATION
00000E BCTBLTYPE              DS   CL1            BLOCK TYPE FOR FUNCTION
       *                                          (VALUE AS IN BCETYPE)
00000F BCTBLSPARE             DS   BL1            SPARE
000010 BCTBLIOHEAD            DS   A              HEAD OF CHAIN FOR BCES
       *                                          WITH I/O OUTSTANDING
000014 BCTBLWTABLE               DS  XL8      TABLE OF WEIGHTS FOR REPL
       * VALUE IN PAREN BELOW IS DEFAULT VALUE LOADED BY SVC OPEN.
00001C                 ORG BCTBLWTABLE
000014 BCTBLWDATA                DS  XL1      DATA BLOCK NO HOLD       (1)
000015 BCTBLWDATAH               DS  XL1      DATA BLOCK HOLD          (2)
000016 BCTBLWINDEX               DS  XL1      INDEX BLOCK (PRIMARY)    (3)
000017 BCTBLWROOT                DS  XL1      INDEX ROOT (PRIMARY)     (5)
000018 BCTBLWADATA               DS  XL1      LOW LEVEL ALT BLOCK      (1)
000019 BCTBLWAINDEX              DS  XL1      INDEX BLOCK (ALT)        (3)
00001A BCTBLWAROOT               DS  XL1      INDEX ROOT (ALT)         (5)
00001B BCTBLWRES                 DS  XL1      RESERVED WEIGHT CLASS    (0)
00001C BCTBLEXPAND            DS   BL4            EXPANSION AREA (BCTBL)
       * END OF BCTBL HEADER; BCE'S BEGIN HERE
000020 BCTBLBCE1              DS   BL56           BUFFER CONTROL ENTRY
000058 BCTBLBCE2              DS   BL56           BUFFER CONTROL ENTRY 2,ETC
```

```
                EXTRD
000000 EXTRD          DSECT
       *
       *        SYMBOLIC DEFINITION OF THE RESULT AREA OF THE 'EXTRACT'
       *        SUPERVISOR ROUTINE, AND ID CODES FOR CLASS 3 AND 4 EXTRACT
       *        ITEMS
       *
       *        DATE 5/27/79
       *        VERSION 2.01       (INCLUDES 2246C WORKSTATION)
       *
000000 EXTRDBEGIN                   DS  0XL1        (UNALIGNED)
       *
       EXTRDIDMAX                   EQU   73        MAX ID # CURRENTLY IN  11↑
       *                                            USE-FROM EXTRACT MACRO 11↑
       ************************CLASS 0*****************************************
000000 EXTRDCLASS0                  DS  0XL12       RETURNED FOR CLASS 0:
000000 EXTRDNRES                    DS  AL4         PHYSICAL MEMORY (BYTES)
       *                                              NOT PERMANENTLY RESIDENT
000004 EXTRDOCNT                    DS  HL2         NUMBER OF FILES WHICH
       *                                              CURRENT TASK MAY HAVE
       *                                              OPEN, EXCLUDING FILES
       *                                              ALREADY OPEN
000006 EXTRDWS                      DS  HL2         TASK'S ASSOCIATED
       *                                              WORKSTATION NUMBER, OR
       *                                              -1 IF NONE
000008 EXTRDSTACK                   DS  AL4         REMAINING STACK SPACE
       *
       ************************CLASS 1*****************************************
00000C            ORG EXTRDBEGIN
000000 EXTRDCLASS1                  DS  0XL98       RETURNED IN ADDITION
       *                                              FOR CLASS 1:
000000            ORG EXTRDBEGIN+L'EXTRDCLASS0
00000C EXTRDDYVAL                   DS  FL4         ONE DAY IN CLOCK UNITS
000010 EXTRDSYSVOL                  DS  0CL6        SYSTEM DEFAULT LIBRARY
000010 EXTRDSCDVOL                  DS  CL6           VOLUME NAME
000016 EXTRDSYSLIB                  DS  0CL8        SYSTEM DEFAULT LIBRARY
000016 EXTRDSCDNAME                 DS  CL8           NAME
00001E EXTRDPRINTER                 DS  0HL2        DEFAULT ONLINE PRINTER
00001E EXTRDDEFPRT                  DS  HL2           DEVICE NUMBER,
       *                                              OR -1 OF NONE
000020 EXTRDRUNVOL                  DS  0CL6
000020 EXTRDUPDVOL                  DS  CL6         USER PROGRAM LIB.VOLUME
000026 EXTRDRUNLIB                  DS  0CL8
000026 EXTRDUPDNAME                 DS  CL8         USER PROGRAM LIB.NAME
00002E EXTRDEXFLGS                  DS  BL4         'EXECUTE' ACCESS MASK
000032 EXTRDINVOL                   DS  0CL6
000032 EXTRDVOL                     DS  CL6         DEFAULT INPUT VOLUME
000038 EXTRDINLIB                   DS  0CL8
000038 EXTRDFILE1                   DS  CL8         DEFAULT INPUT LIBRARY
000040 EXTRDRDFLGS                  DS  BL4         'READ' ACCESS MASK
000044 EXTRDOUTVOL                  DS  0CL6
000044 EXTRDVOL0                    DS  CL6         DEFAULT OUTPUT VOLUME
00004A EXTRDOUTLIB                  DS  0CL8
00004A EXTRDFILE10                  DS  CL8         DEFAULT OUTPUT LIBRARY
000052 EXTRDWTFLGS                  DS  BL4         'WRITE' ACCESS MASK
```

```
000056 EXTRDSEG2BUF                DS  BL2         NUMBER OF SEGMENT 2
       *                                              'BUFFER' PAGES
       *                                              CURRENTLY AVAILABLE
000058 EXTRDPRNTMODE               DS  0CL1
000058 EXTRDPRTTYPE                DS  CL1         PRINT OUTPUT MODE
       *                                              ('S', 'H', OR 'O')
000059 EXTRDFILECLAS               DS  0CL1
000059 EXTRDFPCLASS                DS  CL1         DEFAULT FILE PROTECT
       *                                              CLASS
00005A EXTRDUSERID                 DS  CL3         CURRENT USER LOGON ID
00005D EXTRDTCBSCC                 DS  0HL1        (DO NOT USE)
00005D EXTRDEXTPRIOR               DS  HL1         TASK'S PAGING PRIORITY
00005E EXTRDLINES                  DS  HL1         SUGGESTED LINES/PAGE
00005F EXTRDSPARE1                 DS  0BL3       UNUSED PRIOR TO RELEASE 3.1
       *                                              (WAS BINARY ZEROES)
00005F EXTRDVERSION                DS  XL3         SYSTEM VERSION NUMBER
       *                                              (SEE EXTRDIDVERSION)
       *
       ************************CLASS 2*********************************************
000062             ORG EXTRDBEGIN
000000 EXTRDCLASS2                 DS  0XL8        RETURNED FOR CLASS 2
000000 EXTRDPCPCW                  DS  BL8         PROGRAM OLD PCW FOR
       *                                              LAST PROGRAM CHECK
       *
       ************************CLASS 3*********************************************
       *
       *        FOR CLASS 3, ITEM ID CODES ARE SUPPLIED BY THE EXTRACT SVC
       *        ISSUER AND RETURNED IN INDIVIDUAL AREAS SUPPLIED PER ITEM.
       *        THE FOLLOWING IS A LIST OF ITEM ID CODES. THE LENGTH OF AN
       *        ITEM "ITEMID" MAY BE REFERENCED "L'ITEMID". THE TYPE ATTRIBUTE
       *        MAY BE REFERENCED AS "T'ITEMID".
       *
       ******************************************************************************
       * SYSTEM-WIDE INFORMATION:
       ******************************************************************************
       *
       EXTRDIDNRES                 EQU 0,4,A       PHYSICAL MEMORY (BYTES)
       *                                             NOT PERMANENTLY RESIDENT
       EXTRDIDDYVAL                EQU 4,4,F       ONE DAY IN CLOCK UNITS
       EXTRDIDSYSVOL               EQU 5,6,C       SYSTEM DEFAULT LIBRARY
       *                                             VOLUME NAME
       EXTRDIDSYSLIB               EQU 6,8,C       SYSTEM DEFAULT LIBRARY
       *                                             NAME
       EXTRDIDSYSWORK              EQU 24,8,C      SYSTEM WORK LIBRARY NAME
       *                                             (BACKUP SKIPS)
       EXTRDIDSYSPAGE              EQU 60,8,C      SYSTEM PAGING LIB NAME  1↑
       *                                             (BACKUP SKIPS)          1↑
       EXTRDIDCPU                  EQU 61,2,H      CURRENT CPU ID          1↑
       EXTRDIDHZ                   EQU 62,2,H      A/C LINE FREQUENCY      1↑
       EXTRDIDVERSION              EQU 25,3,X      SYSTEM VERSION NUMBER
       *                                             (PACKED VVRRPP, WHERE
       *                                              'VV' IS VERSION
       *                                              'RR' IS REVISION
       *                                              'PP' IS PATCH LEVEL
       EXTRDIDDEVCNT               EQU 56,4,F      # OF DEVICES ON SYSTEM
```

```
EXTRDIDATOETRT          EQU 57,256,C    ASCII-TO-EBCDIC
*                                       TRANSLATE TABLE
EXTRDIDETOATRT          EQU 58,256,C    EBCDIC-TO-ASCII
*                                       TRANSLATE TABLE
EXTRDCDISKET            EQU 66,2,H      DEVICE # OF SYSTEM'S    06↑
*                                          CENTRAL DISKETTE     06↑
*
*****************************************************************************
* TASK-RELATED INFORMATION:
*****************************************************************************
*
EXTRDIDOCNT             EQU 1,2,H       NUMBER OF FILES WHICH
*                                          CURRENT TASK MAY HAVE
*                                          OPEN, EXCLUDING FILES
*                                          ALREADY OPEN
EXTRDIDWS               EQU 2,2,H       TASK'S ASSOCIATED
*                                          WORKSTATION NUMBER, OR
*                                          -1 IF NONE
EXTRDIDSTACK            EQU 3,4,A       REMAINING STACK SPACE
EXTRDIDEXFLGS           EQU 10,4,B      'EXECUTE' ACCESS MASK
EXTRDIDRDFLGS           EQU 13,4,B      'READ' ACCESS MASK
EXTRDIDWTFLGS           EQU 16,4,B      'WRITE' ACCESS MASK
EXTRDIDUEXFLGS          EQU 63,4,B      USER'S 'EXECUTE' ACCESS 2↑
EXTRDIDURDFLGS          EQU 64,4,B      USER'S 'READ' ACCESS    2↑
EXTRDIDUWTFLGS          EQU 65,4,B      USER'S 'WRITE' ACCESS   2↑
EXTRDIDSEG2BUF          EQU 17,2,H      NUMBER OF SEGMENT 2
*                                          'BUFFER' PAGES
*                                          CURRENTLY AVAILABLE
EXTRDIDUSERID           EQU 20,3,C      CURRENT USER LOGON ID
EXTRDIDUSERNAME         EQU 26,24,C     USER NAME (FROM USERLIST)
EXTRDIDEXTPRIOR         EQU 21,1,H      TASK'S PAGING PRIORITY
EXTRDIDPCPCW            EQU 23,8,X      PROGRAM OLD PCW FOR
*                                          LAST PROGRAM CHECK
EXTRDIDTASK#            EQU 27,4,A      UNIQUE TASK IDENTIFIER
EXTRDIDTASKTYPE         EQU 28,2,C      TASK TYPE:
*                                          'F ' FOR FOREGROUND
*                                          'FS' FOR DEDICATED
*                                               SYSTEM TASK (FG)
*                                          'B ' FOR BACKGROUND
*                                          'BS' FOR DEDICATED
*                                               SYSTEM TASK (BG)
EXTRDIDCURVOL           EQU 29,6,C      VOLUME OF CURRENT PROGRAM
EXTRDIDCURLIB           EQU 30,8,C      LIBRARY OF CURRENT PROGRAM
EXTRDIDWORKLIB          EQU 31,8,C      WORK LIBRARY NAME
*                                          CONTRUCTED FROM USER ID
*                                          OR BG TASK #
EXTRDIDSPOOLIB          EQU 32,8,C      SPOOL LIBRARY NAME
*                                          CONSTUCTED FROM USER ID
*                                          OR BG TASK #
EXTRDIDJOBNAME          EQU 71,8,C      NAME OF BACKGROUND JOB  8↑
EXTRDIDSEG2SIZE         EQU 33,4,F      LENGTH OF SEG 2 IN BYTES
EXTRDIDSTATIC           EQU 34,4,A      ADDRESS OF START OF STATIC
*                                          AREAS (R14 AT PROGRAM
*                                          INVOCATION)
*
```

```
*****************************************************************************
* USER DEFAULTS. MAY BE SET USING SET SVC.
*****************************************************************************
*
EXTRDIDPRINTER              EQU 7,2,H       DEFAULT ONLINE PRINTER
*                                              DEVICE NUMBER,
*                                              OR -1 IF NONE
EXTRDIDRUNVOL               EQU 8,6,C       USER PROGRAM VOLUME
*                                              USED BY CP RUN COMMAND
EXTRDIDRUNLIB               EQU 9,8,C       USER PROGRAM LIBRARY
*                                              USED BY CP RUN COMMAND
EXTRDIDINVOL                EQU 11,6,C      DEFAULT INPUT VOLUME
EXTRDIDINLIB                EQU 12,8,C      DEFAULT INPUT LIBRARY
EXTRDIDOUTVOL               EQU 14,6,C      DEFAULT OUTPUT VOLUME
EXTRDIDOUTLIB               EQU 15,8,C      DEFAULT OUTPUT LIBRARY
EXTRDIDPRNTMODE             EQU 18,1,C      PRINT OUTPUT MODE
*                                              ('S', 'H', 'O', OR 'K')
EXTRDIDFILECLAS             EQU 19,1,C      DEFAULT FILE PROTECT
*                                              CLASS
EXTRDIDLINES                EQU 22,1,H      SUGGESTED LINES/PAGE
EXTRDIDPROGVOL              EQU 35,6,C      USER PROGRAM VOLUME
*                                              USED BY LINK SVC
EXTRDIDPROGLIB              EQU 36,8,C      USER PROGRAM LIBRARY
*                                              USED BY LINK SVC
EXTRDIDWORKVOL              EQU 37,6,C      DEFAULT WORK VOLUME
EXTRDIDSPOOLVOL             EQU 38,6,C      DEFAULT SPOOL VOLUME
EXTRDIDPRTCLASS             EQU 39,1,C      DEFAULT PRINT CLASS FOR
*                                              PRINT FILES (A-Z)
EXTRDIDFORM#                EQU 40,1,H      DEFAULT FORM NUMBER FOR
*                                              PRINT FILES (0-254)
EXTRDIDJOBQUEUE             EQU 68,1,C      DEFAULT JOB STATUS      8↑
*                                              ('R' OR 'H')            8↑
EXTRDIDJOBCLASS             EQU 69,1,C      DEFAULT JOB CLASS       8↑
*                                              ('A' TO 'Z')            8↑
EXTRDIDJOBLIMIT             EQU 70,4,F      DEFAULT JOB CPU TIME    8↑
*                                              LIMIT (SECONDS)         8↑
*
*****************************************************************************
* RUN STATISTICS
*****************************************************************************
*
EXTRDIDWSIO                 EQU 41,4,F      COUNT OF WORKSTATION IOS
*                                              THIS RUN
EXTRDIDTAPEIO               EQU 42,4,F      COUNT OF TAPE IOS THIS RUN
EXTRDIDDISKIO               EQU 43,4,F      COUNT OF DISK IOS THIS RUN
EXTRDIDPRINTIO              EQU 44,4,F      COUNT OF PRINTER IOS
EXTRDIDOTIO                 EQU 45,4,F      COUNT OF OTHER IOS
EXTRDIDPICOUNT              EQU 46,4,F      PROGRAM PAGEIN COUNT
EXTRDIDPOCOUNT              EQU 47,4,F      PROGRAM PAGEOUT COUNT
EXTRDIDSICOUNT              EQU 48,4,F      SYSTEM PAGEIN COUNT
EXTRDIDSOCOUNT              EQU 49,4,F      SYSTEM PAGEOUT COUNT
EXTRDIDETIME                EQU 50,4,F      ELAPSED TIME OF RUN SINCE
*                                              COMMAND PROCESSOR
*                                              INITIATION, IN
*                                              HUNDREDTHS OF SECONDS
```

```
        EXTRDIDPTIME                EQU 51,4,F      PROCESSOR TIME OF RUN
        *                                              SINCE COMMAND PROCESSOR
        *                                              INITIATION, IN
        *                                              HUNDREDTHS OF SECONDS
        *
        ************************CLASS 4*****************************************
        *
        *       CLASS 4 ITEMS ARE SIMILAR TO CLASS 3 ITEMS, EXCEPT THAT
        *       ADDITIONAL INPUT IS REQUIRED PER ITEM.
        *
        EXTRDIDDEVICE               EQU 52,24,B
        *   INPUT  = DEVICE ADDRESS (1 BYTE)
        *   OUTPUT AS FOLLOWS:
000008          ORG EXTRDBEGIN
000000  EXTRDDEVCLASS               DS  HL1         DEVICE CLASS:
        EXTRDDEVCLASSWS             EQU 1              WORKSTATION          9↑
        EXTRDDEVCLASSMT             EQU 2              MAGNETIC TAPE        9↑
        EXTRDDEVCLASSDK             EQU 3              DISK                 9↑
        EXTRDDEVCLASSPR             EQU 4              PRINTER              9↑
        EXTRDDEVCLASSTC             EQU 5              TELECOMMUNICATIONS   9↑
000001  EXTRDTYPE                   DS  HL1         DEVICE TYPE:
        EXTRDTYPE2246P              EQU 017         2246P   WORKSTATION      9↑
        EXTRDTYPE2246S              EQU 018         2246S   WORKSTATION      9↑
        EXTRDTYPE2246R              EQU 019         2246R   WORKSTATION      9↑
        EXTRDTYPE2246C              EQU 020         2246C   WORKSTATION      9↑
        EXTRDTYPE2246K              EQU 021         2246K   WORKSTATION    0↑9↑
        EXTRDTYPE2266C              EQU 022         ARCHIVER C W/S        3A↑9↑
        EXTRDTYPE2266S              EQU 023         ARCHIVER S W/S        3A↑9↑
        EXTRDTYPE2246SI             EQU 024         IDEOGRAPHIC S W/S   7↑9↑14↑
        EXTRDTYPE2246D              EQU 025         IBM 029 S W/S          10↑
        EXTRDTYPE2256C              EQU 026         64K C W/S              12↑
        EXTRDTYPE2276C              EQU 027         ARCHIVER C 64K W/S     12↑
        EXTRDTYPE2246O              EQU 028         OKIDATA WORKSTATION    12↑
        EXTRDTYPE2246CI             EQU 029         IDEOGRAPHIC C W/S      14↑
        EXTRDTYP2246SIK             EQU 030         IDEOGRAPHIC/K S W/S    14↑
        EXTRDTYPE2246RK             EQU 031         REMOTE KATAKANA W/S    14↑
        *
        EXTRDTYPE8021               EQU 033         8021  MAG TAPE (800 BPI)9↑
        EXTRDTYPE2209V              EQU 034         2209V MAG TAPE         9↑
        *                                              (1600 BPI)
        EXTRDTYPE2209V2             EQU 035         2209V-2 MAG TAPE       9↑
        *                                              (800/1600 BPI)
        EXTRDTYPE2209V3             EQU 036         2209V-3 7-TRACK MAG TAPE9↑
        EXTRDTYPE2260V              EQU 050         2260V  DISK (408CYL F/R)9↑
        EXTRDTYPE2265V1             EQU 051         2265V-1 DISK(823CYL REM)9↑
        EXTRDTYPE2265V2             EQU 052         2265V-2 DISK(823CYL REM)9↑
        EXTRDTYPE2270V              EQU 053         2270V   DISKETTE       9↑
        *                                              (77CYL REM)
        EXTRDTYPE2280V1             EQU 054         2280V-1 DISK(823CYL F/R)9↑
        EXTRDTYPE2280V2             EQU 055         2280V-2 DISK(823CYL F/R)9↑
        EXTRDTYPE2280V3             EQU 056         2280V-3 DISK(823CYL F/R)9↑
        EXTRDTYPE2270V1             EQU 057         2270V-1 DISKETTE     03↑9↑
        *                                           (HARD SECTORED)        03↑
        EXTRDTYPE2270V2             EQU 058         2270V-2 DISKETTE     03↑9↑
        *                                           (SOFT SECTORED)        03↑
```

```
EXTRDTYPE2270V3          EQU 059        2270V-3 DISKETTE          03↑9↑
*                                       (HARD OR SOFT SECTORED)   03↑
EXTRDTYPE9614            EQU 060        9614 FIXED DISK           4↑9↑
*
EXTRDTYPE2221V           EQU 065        2221V   PRINTER           9↑
*                                       (200CPS MAT)
EXTRDTYPE2231V2          EQU 067        2231V-2 PRINTER           9↑
*                                       (120CPS MAT)
EXTRDTYPE2261V           EQU 068        2261V   PRINTER           9↑
*                                       (240LPM MAT)
EXTRDTYPE2263V1          EQU 069        2263V-1 PRINTER           9↑
*                                       (300LPM TR)
EXTRDTYPE2263V2          EQU 070        2263V-2 PRINTER           9↑
*                                       (600LPM TR)
EXTRDTYPE8047            EQU 071        8047    PRINTER           9↑
*                                       (225LPM TR)
EXTRDTYPE8048            EQU 072        8048    PRINTER           9↑
*                                       (450LPM TR)
EXTRDTYPE2281V           EQU 073        2281V   PRINTER (30CPS    9↑
*                                         DAISY WHEEL)
EXTRDTYPE2231V6          EQU 074        2231V-6 PRINTER           9↑
*                                       (120 CPS MATRIX)
EXTRDTYPE2263V3          EQU 075        2263V-3 PRINTER           9↑
*                                       (430 LPM TR)
EXTRDTYPE2273V1          EQU 076        2273V-1 PRINTER           0↑9↑
*                                       (REMOTE)                  0↑
***                                       ***
*** (SERIAL PRINTERS -- TENTATIVE PRODUCT NUMBERS ) ***
***                                       ***
EXTRDTYPE2221VS          EQU 097        2221V-S PRINTER           9↑
*                                       (200 CPS MATRIX)
EXTRDTYP2231V2S          EQU 099        2231V-2S PRINTER          9↑
*                                       (120 CPS MATRIX)
EXTRDTYPE2261VS          EQU 100        2261V-S PRINTER           9↑
*                                       (240 LPM MATRIX)
EXTRDTYP2263V1S          EQU 101        2263V-1S PRINTER          9↑
*                                       (300 LPM TR)
EXTRDTYP2263V2S          EQU 102        2263V-2S PRINTER          9↑
*                                       (600 LPM TR)
EXTRDTYPE2281VS          EQU 105        2281V-S PRINTER (30CPS    9↑
*                                         DAISY WHEEL)
EXTRDTYPE6581W           EQU 105        6581W PRINTER             0↑9↑
*                                       ( 30 CPS DAISY)           0↑
EXTRDTYP2231V6S          EQU 106        2231V-6S PRINTER          9↑
*                                       (120 CPS MATRIX)
EXTRDTYP2263V3S          EQU 107        2263V-3S PRINTER          9↑
*                                       (430 LPM TR)
EXTRDTYPE6581WC          EQU 108        6581-WC WIDE PRINTER      0↑9↑
*                                       ( 40 CPS DAISY )          0↑
EXTRDTYPE5573            EQU 109        5573 PRINTER              0↑9↑
*                                       (300 LPM BAND)            0↑
EXTRDTYPE5574            EQU 110        5574 PRINTER              0↑9↑
*                                       (600 LPM BAND)            0↑
EXTRDTYPE5521K           EQU 111        5521K KATAKANA PRINTER    4↑9↑
*                                       (200 CPS MATRIX)          4↑
```

```
        EXTRDTYPE55312K             EQU 112         5531-2K KATAKANA PRT  4↑9↑
        *                                              (120 CPS MATRIX)     4↑
        EXTRDTYPE5548Z              EQU 113         5548Z TYPESETTER      4↑9↑
        *
        EXTRDTYPEIP41D              EQU 114         INTELLIGENT IMAGE PRT 5↑9↑
        *
        EXTRDTYPE5521I              EQU 115         IDEOGRAPHIC MAT   PRT 7↑9↑
        *
        EXTRDTYPE5581WD             EQU 116         DUAL-HEAD DAISY PRT   12↑
        *                                                                 12↑
        EXTRDTYPECP210              EQU 117         OKIDATA MATRIX PRT    12↑
        *                                                                 12↑
        EXTRDTYPE5521IK             EQU 118         IDEOGRAPHIC/K MAT PRT 14↑
        *                                                                 14↑
        EXTRDTYPE1022S              EQU 119         180 CPS MAT PRT       14↑
        *                                                                 14↑
        ***
        EXTRDTYPETC                 EQU 081         BATCH TC DEVICE        9↑
        *
000002  EXTRDDEVUSAGE               DS  CL2         DEVICE USAGE:
        EXTRDDEVUEX                 EQU C'EX'          EXCLUSIVE USE      06↑
        EXTRDDEVUSH                 EQU C'SH'          SHARED USE         06↑
        EXTRDDEVUDT                 EQU C'DT'          DETACHED           06↑
000004  EXTRDDEVUSER                DS  AL4         TASK IDENTIFIER OF
        *                                              CURRENT DEVICE OWNER,
        *                                              -1 IF NONE
000008  EXTRDDEVREM                 DS  CL6         VOLSER OF REMOVABLE VOLUME
        *                                              DEFINED ONLY FOR DISK AND
        *                                              TAPE. CL6' ' IF NOTHING
        *                                              MOUNTED.
00000E  EXTRDDEVFIXED               DS  CL6         VOLSER OF FIXED VOLUME
        *                                              DEFINED ONLY FOR DISK.
        *                                              CL6' ' IF NOTHING MOUNTED.
000014  EXTRDDEVSPARE               DS  CL4         (UNUSED)
        *
        ********************************************************************************
        *
        EXTRDIDVOLUME               EQU 53,24,B
        *   INPUT  = VOLUME SERIAL NUMBER (6 BYTES)
        *   OUTPUT AS FOLLOWS:
000018          ORG EXTRDBEGIN
000000  EXTRDVOLDEV                 DS  AL1         DEVICE NUMBER, OR -1 IF
        *                                              VOLUME NOT MOUNTED
        * NOTE: EXTRDVOLTYPE, EXTRDVOLLABEL,
        *       EXTRDVOLUSAGE, AND EXTRDVOLUSERID
        *       ARE ALL BLANK IF VOLUME IS NOT MOUNTED.
000001  EXTRDVOLTYPE                DS  CL1         VOLUME TYPE:
        EXTRDVOLTYPER               EQU C'R'           REMOVABLE
        EXTRDVOLTYPEF               EQU C'F'           FIXED
000002  EXTRDVOLLABEL               DS  CL2         LABEL TYPE:
        EXTRDVOLLABSL               EQU C'SL'          STANDARD LABEL     06↑
        EXTRDVOLLABNL               EQU C'NL'          NO LABEL           06↑
000004  EXTRDVOLUSAGE               DS  CL2         VOLUME USAGE:
        EXTRDVOLUSH                 EQU C'SH'           SHARED USE        06↑
        EXTRDVOLURR                 EQU C'RR'           RESTRICTED..      06↑
```

```
       *                                                    ..REMOVAL      06↑
       EXTRDVOLUPR                  EQU C'PR'               PROTECTED USE  06↑
       EXTRDVOLUEX                  EQU C'EX'               EXCLUSIVE USE  06↑
000006 EXTRDVOLUSER                 DS  AL4          TASK IDENTIFIER OF VOLUME
       *                                               MOUNTER, -1 IF NONE
00000A EXTRDVOLBC                   DS  HL2          BLOCKS PER CYLINDER
00000C EXTRDVOLMAXTFR               DS  HL2          MAXIMUM TRANSFER IN BYTES
00000E EXTRDVOLCV                   DS  HL2          CYLINDERS PER VOLUME
000010 EXTRDVOLCVP                  DS  HL2          CYLINDERS PER PHYSICAL
       *                                               VOLUME, INCLUDING BAD
       *                                               AND UNUSED BLOCKS
000012 EXTRDVOLOCNT                 DS  HL2          NUMBER OF FILES OPEN
000014 EXTRDVOLSECT                 DS  CL1          SECTOR TYPE:           3B↑
       *                                               -- DISKETTE ONLY --   3B↑
       EXTRDVOLSECTS                EQU C'S'            SOFT SECTOR   3B↑  06↑
       EXTRDVOLSECTH                EQU C'H'            HARD SECTOR   3B↑  06↑
000015 EXTRDVOLADDR                 DS  CL1          ADDRESSING IN EFFECT:  3B↑
       *                                               -- DISKETTE ONLY --   3B↑
       EXTRDVOLADDRN                EQU C'N'            NON-STANDARD  3B↑  06↑
       EXTRDVOLADDRS                EQU C'S'            STANDARD      3B↑  06↑
000016 EXTRDVOLSPARE                DS  BL2          (UNUSED)               03↑
       *
       ********************************************************************************
       *
       EXTRDIDOTASK                 EQU 54,48,B
       *   INPUT  = TASK IDENTIFIER (4 BYTES)
       *   OUTPUT AS FOLLOWS:
000018             ORG EXTRDBEGIN
000000 EXTRDOTASKWS                 DS  AL1          WORKSTATION DEVICE NUMBER
       *                                               OF TASK SPECIFIED,
       *                                               OR -1 IF NOT FOREGROUND
       *                                               TASK
000001 EXTRDOTASKUID                DS  CL3          CURRENT USER ID FOR TASK
       *                                               SPECIFIED, OR BLANK
000004 EXTRDOTASKNAME               DS  CL24         CURRENT USER NAME FOR TASK
       *                                               SPECIFIED, OR BLANK
00001C EXTRDOTASKTYPE               DS  CL2          TASK TYPE -
       *                                                 SEE EXTRDIDTASKTYPE
00001E EXTRDOTASKSPARE              DS  BL18         (UNUSED)
       *
       ********************************************************************************
       *
       EXTRDIDTAPEVOL               EQU 55,20,B
       *   INPUT  = VOLUME SERIAL NUMBER (6 BYTES)
       *   OUTPUT AS FOLLOWS:
000030             ORG EXTRDBEGIN
000000 EXTRDTAPEDEV                 DS  AL1          DEVICE NUMBER, OR -1 IF
       *                                               VOLUME NOT MOUNTED
000001 EXTRDTAPESPAR1               DS  BL1          (UNUSED)
       *
       * NOTE: EXTRDTAPELABEL,
       *       EXTRDTAPEUSAGE, AND EXTRDTAPEUSER ARE ALL BLANK
       *       IF NO TAPE MOUNTED.
000002 EXTRDTAPEDEN                 DS  HL2          TAPE DENSITY IN BINARY
       *                                               BPI (556,800 OR 1600)
```

```
000004 EXTRDTAPELABEL              DS  CL2        LABEL TYPE:
       EXTRDTAPELABAL              EQU C'AL'        ANSI LABEL          06↑
       EXTRDTAPELABNL              EQU C'NL'        NO LABEL            06↑
       EXTRDTAPELABIL              EQU C'IL'        IBM LABEL           06↑
000006 EXTRDTAPEUSAGE              DS  CL2        VOLUME USAGE:
       EXTRDTAPEUSH                EQU C'SH'        SHARED USE          06↑
       EXTRDTAPEUEX                EQU C'EX'        EXCLUSIVE USE       06↑
000008 EXTRDTAPEUSER               DS  AL4        TASK IDENTIFIER OF VOLUME
       *                                             MOUNTER, -1 IF NONE
00000C EXTRDTAPEFSEQ               DS  HL2        FILE SEQUENCE NUMBER
00000E EXTRDTAPESPAR2              DS  BL6        (UNUSED)
       *
       ********************************************************************************
       *
       EXTRDIDDEVLIST              EQU 59,3,B
       *   INPUT  = DEVICE CLASS, AS IN EXTRDDEVCLASS (1 BYTE)
       *   OUTPUT AS FOLLOWS:
000014           ORG EXTRDBEGIN
000000 EXTRDDLTOT                  DS  HL1        TOTAL NUMBER OF DEVICES IN
       *                                             SPECIFIED CLASS
000001 EXTRDDLNUM                  DS  HL1        NUMBER OF DEVICES
       *                                             ADDRESSES SUPPLIED
000002 EXTRDDLIST                  DS  0X         DEVICE LIST
000002 EXTRDDLENTRY                DS  AL1        DEVICE ADDRESS, OR X'FF'
       *                                             IF NO MORE DEVICES
       ********************************************************************************
       *
       *                  TC RELATED INFORMATION                              11↑
       EXTRDIDDLPNAME              EQU 72,38,B                                 11↑
       *   INPUT = DLPNAME (4 BYTE CHAR. STRING)                              11↑
       *   OUTPUT AS FOLLOWS:                                                 11↑
       *                                                                      11↑
000003         ORG   EXTRDBEGIN                                               11↑
000000 EXTRDDLPDEVMAP              DS  XL4        BITMAP OF DEVS ON DLP       11↑
000004 EXTRDDLPDEV#1               DS  XL2        1ST DEV ON DLP              11↑
000006 EXTRDDLPTYPE                DS  XL1        DLP TYPE                    11↑
       EXTRDDLPTYPE1               EQU 1          TYPE 22V06-1                11↑
       EXTRDDLPTYPE2               EQU 2          TYPE 22V06-2                11↑
       EXTRDDLPTYPE3               EQU 3          TYPE 22V06-3                11↑
000007 EXTRDDLPLINECNT             DS  XL1        # OF LINES CONTROLLABLE     11↑
       *                                           BY DLP                      11↑
000008 EXTRDMCSTATUS               DS  XL1        MICROCODE FILE STATUS       11↑
       *                                           0 - IF STOPPED              11↑
       *                                           HI BIT ON IF LOADED         11↑
000009 EXTRDDLPSPARE               DS  XL3        RESERVED FOR FUTURE USE     11↑
00000C EXTRDMCFILE                 DS  XL8        MICROCODE FILE NAME         11↑
000014 EXTRDMCLIB                  DS  XL8        MICROCODE LIB  NAME         11↑
00001C EXTRDMCVOL                  DS  XL6        VOLUME NAME FOR MCFILE      11↑
       *                                                                      11↑
000022 EXTRDDLPRSRV                DS  XL1        RESERVATION STATUS-DLP      11↑
       *                                           HI BIT ON IF RESERVED       11↑
000023 EXTRDDLPTASK#               DS  XL3        RESERVING TASK #      13↑11↑
000026         ORG   ,                                                         11↑
```

```
        ********************************************************************************11↑
        *                                                                               11↑
        EXTRDIDDLPDEV#              EQU 73,8,B                                          11↑
        *   INPUT = DEVICE NUMBER (2 BYTES)                                             11↑
        *   OUTPUT AS FOLLOWS:                                                          11↑
        *                                                                               11↑
000062          ORG   EXTRDBEGIN                                                        11↑
000000 EXTRDDEVSTATUS               DS  XL1        DEV RESERVATION STATUS               11↑
        EXTRDDEVRSRV                EQU X'40'      DEVICE RESERVED                      11↑
        EXTRDDEVOPEN                EQU X'80'      DEVICE OPEN                          11↑
000001 EXTRDDEVTASK#                DS  XL3        RESERVING TASK #              13↑11↑
000004 EXTRDDEVDLPNAME              DS  XL4        DLPNAME FOR DEVICE                   11↑
000008          ORG   ,                                                                 11↑
        ********************************************************************************
        *
        *                  DEVICE CLUSTER INFORMATION                                 06↑
        EXTRDIDCLUSTER              EQU 67,16,B                                         06↑
        *   INPUT  = DEVICE NUMBER (2 BYTES)                                            06↑
        *   OUTPUT AS FOLLOWS:                                                          06↑
000062           ORG EXTRDBEGIN                                                         06↑
000000 EXTRDADISKET                 DS  HL2          DEVICE # OF ASSOCIATED             06↑
        *                                               ARCHIVER DISKETTE,            06↑
        *                                               OR ZERO IF NONE               06↑
000002 EXTRDSPARE                   DS  BL14         (UNUSED)                           06↑
        *
000010          ORG
```

```
                FDR1
000000 FDR1          DSECT
       *
       *        THE FORMAT 1 FILE DESCRIPTOR RECORD (FDR1) DESCRIBES THE
       *        ATTRIBUTES OF A FILE, INCLUDING THE FIRST THREE EXTENTS
       *        OF THE FILE. EVERY FILE ON A VOLUME (EXCEPT THE VTOC
       *        AND VOLUME LABEL/IPL TEXT AREA) HAS A FORMAT 1 FDR
       *        ASSOCIATED WITH IT. FORMAT 1 FDRS ARE LOCATED THROUGH THE
       *        FDX1 AND FDX2 BLOCKS. THERE ARE UP TO 25 80-BYTE
       *        FDR RECORDS PER VTOC BLOCK. THE 2045TH BYTE OF A BLOCK
       *        CONTAINING FDRS CONTAINS AN ASCII 'F'. ALL BLOCKS CONTAINING
       *        AVAILABLE 80-BYTE SLOTS FOR
       *        FDRS ARE CHAINED TOGETHER BY BLOCK NUMBERS (WITHIN VTOC,
       *        FROM 0) IN THE 2047TH AND 2048TH BYTES OF EACH SUCH
       *        BLOCK, EXACTLY AS ARE THE FDX2 BLOCKS.
       *        THE NUMBER OF AVAILABLE 80-BYTE SLOTS IN A BLOCK
       *        IS MAINTAINED IN BINARY IN THE 2043TH AND 2044TH
       *        BYTES OF THE BLOCK.
       *
       *        DATE 5-17-77
       *        VERSION 2.02 (UPDATED FOR ALTERNATE INDEXING)
       *
000000 FDR1BEGIN                DS  0F
000000 FDR1FORMAT               DS  CL1         FORMAT OF FDR (ASCII '1')
       *                                ('N' FOR FDR RECORD NOT IN USE)
       FDR1INUSE                EQU C'1'       FDR1 IN USE                 1↑
       FDR1NOTUSED              EQU C'N'       FDR1 NOT IN USE             1↑
000001 FDR1XTNTCOUNT            DS  BL1        COUNT OF EXTENTS IN USE
000002 FDR1ORG                  DS  BL1        FILE ORGANIZATION
       FDR1ORGCONSEC            EQU X'01'      CONSECUTIVE ORGANIZATION
       FDR1ORGINDEXED           EQU X'02'      INDEXED ORGANIZATION
       FDR1ORGWP                EQU X'04'      WORD PROCESSING FILE
       FDR1ORGVLEN              EQU X'20'      VARIABLE-LENGTH RECORDS
       FDR1ORGPRINT             EQU X'40'      PRINT FILE
       FDR1ORGPROG              EQU X'80'      PROGRAM FILE
000003 FDR1FLAGS                DS  BL1        FLAGS FOR STATUS
       FDR1FLAGSUPDAT           EQU X'80'      SET TO 0 BY CREATFDR,
       *                                       SET TO 1 BY UPDATFDR
       FDR1FLAGSCOMP            EQU X'40'      COMPRESSED RECORDS
       FDR1FLAGSRECOV           EQU X'20'      USE PREFORMAT AND RECOVERY
       *                                       PROCEDURES FOR THIS FILE
       FDR1FLAGSALTX            EQU X'10'      INDEXED FILE HAS AN AXD1
       *                                       BLOCK AND ALT-INDICES IF SET
       FDR1FLAGSLOG             EQU X'08'      CONSEC LOG FILE FLAG
       FDR1FLAGSPART            EQU X'04'      PARTIAL BACKUP FILE
       FDR1FLAGSADMS            EQU X'02'      ADMS FILE
       FDR1FLAGSPRIV            EQU X'01'      PROGRAM FILE CARRIES
       *                                       ADDITIONAL ACCESS PRIVILEGES
000004 FDR1X1PTR                DS  H          FDX1 BLOCK * 169 + FDX1
       *                                       ITEM IN BLOCK (FROM 0)000006
FDR1FILENAME    DS          CL8       MEMBER NAME
00000E FDR1FILESECTION          DS  CL1        VOLUME IN A MULTI-VOLUME
       *                                       FILE (ALWAYS ASCII '1')
00000F FDR1CREDATE              DS  PL3        CREATION DATE (PACKED YYDDD+
000012 FDR1MODDATE              DS  PL3        LAST MODIFICATION DATE
```

```
      *                                          (PACKED YYDDD+)
000015 FDR1EXPDATE              DS  PL3          EXPIRATION DATE (PACKED YYDD
000018 FDR1FPCLASS              DS  CL1          FILE PROTECTION ACCESS-CLASS
000019 FDR1CREATOR              DS  CL3          USER LOGON IDENTIFICATION OF
      *                                          FILE CREATOR
00001C FDR1BLKSIZE              DS  H            PHYSICAL BLOCK SIZE (2048)
00001E FDR1SECEXT               DS  H            NO. BLOCKS SECONDARY EXTENT
000020 FDR1X1STRT               DS  FL3          PRIMARY EXTENT START BLOCK
000023 FDR1X1END                DS  FL3          PRIMARY EXTENT END BLOCK + 1
000026 FDR1X2STRT               DS  FL3          2ND EXTENT START
000029 FDR1X2END                DS  FL3          2ND EXTENT END
00002C FDR1X3STRT               DS  FL3          3RD EXTENT START
00002F FDR1X3END                DS  FL3          3RD EXTENT END
      ********************************************************************
      * ORGANIZATION-DEPENDENT SECTION:
      ********************************************************************
000032 FDR1SPARE2               DS  BL2          (UNUSED FOR CONSECUTIVE
      *                                          FILES)
000034 FDR1NRECS                DS  F            NUMBER OF DATA RECORDS
000038 FDR1RECSIZE              DS  H            LOGICAL RECORD SIZE
00003A FDR1SPARE3               DS  BL1          (UNUSED UNLESS
      *                                          FDR1FLAGSALTX SET)
00003B FDR1EBLK                 DS  FL3          LAST RECORD'S BLOCK WITHIN
      *                                          FILE
00003E FDR1EREC                 DS  H            LAST RECORD'S NUMBER IN LAST
      *                                          BLOCK FOR CONSECUTIVE FILES
      *                                          WITH FIXED-LENGTH RECORDS
      *                                          (FOR INDEXED FILES, NUMBER
      *                                          OF PRIMARY INDEX LEVELS)
000040 FDR1SPARE4               DS  BL12         (UNUSED FOR CONSECUTIVE
      *                                          FILES WHICH ARE NOT PROGRAM
      *                                          FILES)
      ********************************************************************
      * FOR WORD PROCESSING FILES ONLY:
      ********************************************************************
00004C                          ORG FDR1SPARE2
000032 FDR1WPBLKSIZE            DS  XL1          WP FILE BLOCK SIZE
000033 FDR1WPBLS                DS  XL1          BYTES IN LAST
      *                                          SECTOR
      ********************************************************************
      * FOR PROGRAM FILES ONLY:
      ********************************************************************
000034                          ORG FDR1SPARE4
000040 FDR1ACFLAGS              DS  0BL12        ADDITIONAL ACCESS
      *                                          PRIVILEGES:
000040 FDR1WTFLAGS              DS  BL4          ADDITIONAL WRITE
      *                                          PRIVILEGES
000044 FDR1RDFLAGS              DS  BL4          ADDITIONAL READ
      *                                          PRIVILEGES
000048 FDR1EXFLAGS              DS  BL4          ADDITIONAL EXECUTE
      *                                          PRIVILEGES
      ********************************************************************
      * FOR INDEXED FILES ONLY (FILEORG X'02'):
      ********************************************************************
00004C                          ORG FDR1SPARE2
```

```
000032 FDR1PKI          DS  HL1         PACKING FACTOR FOR INDEX
       *                                ITEMS
000033 FDR1PKD          DS  HL1         PACKING FACTOR FOR DATA
       *                                RECORDS
000034                  ORG FDR1SPARE3
00003A FDR1ALTCNT       DS  HL1         NUMBER OF ALTERNATE INDEX
       *                                STRUCTURES DEFINED IN THE
       *                                AXD1-BLOCK (UNUSED UNLESS
       *                                FDR1FLAGSALTX SET)
00003B                  ORG FDR1SPARE4
000040 FDR1KEYPOS       DS  H           PRIMARY KEY POSITION IN
       *                                DATA RECORD
000042 FDR1KEYSIZE      DS  HL1         PRIMARY KEY LENGTH IN BYTES
000043 FDR1HXBLK        DS  FL3         BLOCK-IN-FILE OF ROOT BLOCK
       *                                OF PRIMARY INDEX
000046 FDR1DABLK        DS  FL3         BLOCK-IN-FILE OF STARTING
       *                                BLOCK OF AVAILABLE-BLOCK
       *                                CHAIN
000049 FDR1PTRD         DS  FL3         FIRST DATA BLOCK IN FILE
       *                                (PRIMARY KEY SEQUENCE)
       ********************************************************************
       * FDR CHAIN - IN ALL FDR RECORDS:
       ********************************************************************
00004C FDR1CHAIN        DS  F (HL1,FL3) ADDRESS OF A FORMAT 2 FDR
       *                        FOR THIS FILE'S ADDITIONAL EXTENTS,
       *                        OR A FORMAT 3 FDR FOR THIS FILE'S
       *                        ALTERNATE INDEXING DESCRIPTIONS,
       *                        OR BINARY ZEROES. THE ADDRESS IS
       *                        IN THE FORM:
       *                        (HL1) NUMBER STARTING FROM 0
       *                        OF FDR IN 1-PAGE BLOCK
       *                        (FL3) BLOCK NUMBER IN VTOC FROM 0
       FDR1END          EQU *
       FDR1LENGTH       EQU FDR1END-FDR1BEGIN
       FDR1CNT          EQU 25  # OF FDR1 RECORDS PER BLOCK        1↑
```

```
                FDR2
000000 FDR2          DSECT
       *
       *         THE FORMAT 2 FILE DESCRIPTOR RECORD (FDR2) DESCRIBES UP TO
       *         TEN (10) ADDITIONAL EXTENTS FOR A FILE (BEYOND THE FIRST
       *         THREE). IT IS CHAINED FROM THE FILE'S FORMAT 1 FILE
       *         DESCRIPTOR RECORD. A FORMAT 2 FDR MAY BE CHAINED TO ANOTHER
       *         FORMAT 2 FDR.
       *
       *         DATE 3/28/79
       *         VERSION 4.00
000000 FDR2BEGIN                DS  0F          FULLWORD ALIGNMENT
000000 FDR2FORMAT               DS  CL1         FORMAT (ASCII '2')
000001 FDR2SPARE1               DS  CL5         (UNUSED)
000006 FDR2FILENAME             DS  CL8         FILE NAME AS IN FORMAT 1 FDR
00000E FDR2SPARE2               DS  CL2         (UNUSED)
000010 FDR2X4STRT               DS  FL3         EXTENT 4 (OR 14, 24, ETC.)
       *                            STARTING BLOCK ON VOLUME (FROM 0)
000013 FDR2X4END                DS  FL3         EXTENT 4 ENDING BLOCK ON VOL
000016 FDR2X5TOX13              DS  18FL3       EXTENT DEFINITIONS 5 TO 13
00004C FDR2CHAIN                DS  F (HL1,FL3) CHAIN TO NEXT FORMAT 2 FDR
       *                            FOR ADDITIONAL EXTENTS
       *                            (SEE FDR1CHAIN)
       FDR2END                  EQU *
       FDR2LENGTH               EQU FDR2END-FDR2BEGIN
```

```
                IORE
000000 IORE          DSECT
       *        THE I/O REQUEST ELEMENT (IORE) IS PASSED TO THE SYSTEM'S
       *        START I/O ROUTINE TO INITIATE AN OPERATION, IS QUEUED BY
       *        THAT ROUTINE TO REPRESENT A REQUEST FOR PHYSICAL I/O SERVICE,
       *        AND CONTAINS THE I/O STATUS WORD ON COMPLETION OF THE
       *        REQUEST. IORES MUST BE RESIDENT IN THE SYSTEM
       *        AREA. TASK CONTROL BLOCKS CONTAIN IORES FOR PAGING.
       *        OFBS ADDRESS IORES FOR OTHER I/O.
       *
       *        DATE  3/28/79
       *        VERSION 4.00
       *
000000 IOREBEGIN                DS  0F
000000 IOREREQF                 DS  BL1       REQUEST FLAGS
       IOREREQFNOUNFIX          EQU X'80'     SVC XIO DID NOT TEMPORARILY
       *                                      FIX THIS PAGE, SO ISR MUST
       *                                      NOT UNFIX IT
       IOREREQFPAGEIN           EQU X'40'     THIS IORE IS BEING USED
       *                                      FOR DEMAND PAGEIN
       IOREREQFPAGEOUT          EQU X'20'     THIS IORE IS BEING USED
       *                                      FOR DEMAND PAGEOUT
       IOREREQFIOACT            EQU X'10'     IORE ON PHYSICAL I/O
       *                                      QUEUE
       IOREREQFNOCHK            EQU X'08'     XIO ISSUED; SUCCEEDING
       *                                      CHECK NOT ISSUED
       IOREREQFHALTQ            EQU X'04'     'HALT I/O QUEUE' OPTION OF
       *                                      XIO PASSED HERE TO ISR
       IOREREQFIVRQ             EQU X'02'     'INTERVENTION REQUIRED'
       *                                      INTERRUPT RECEIVED, BUT
       *                                      NOT YET NOTICED BY CHECK SVC
       IOREREQFCIO              EQU X'01'     ISSUE CIO INSTRUCTION IF ON
       *                                      (ELSE ISSUE SIO INSTRUCTION)
000001                     ORG IOREREQF
000000 IORECHN                  DS  A         CHAIN (FROM UCB) IN 3 LOW BY
000004 IOREIOSW                 DS  CL8       I/O STATUS WORD SAVE AREA
00000C IOREIOCW                 DS  CL9       I/O CONTROL WORD SUPPLIED BY
000015                     ORG *-1
000014 IORESEMA                 DS  A         ADDRESS OF COMPLETION
       *                                      SEMAPHORE
       *                                      OR PAGING CODE
       *
       *
       *
       * FIELDS PRESENT IN NON-PAGING IORES ONLY:
       *
000018 IOREIALCNT               DS  HL1       COUNT OF IAL SLOTS
000019 IOREIALUSED              DS  HL1       NUMBER OF IAL SLOTS USED BY
       *                                      THIS IO OPERATION
00001A IORESCC                  DS  HL1       SCAN CLASS FOR PAGE AFTER
       *                                      'UNFIX' ON I/O COMPLETION
00001B IOREWSLEVEL              DS  HL1       STORE AID CHARACTER ON THIS
       *                                      WORKSTATION LEVEL AFTER IO
       *                                      COMPLETION
```

```
00001C IOREIAL                 DS  0A          INDIRECT ADDRESS LIST
       *                                       IF IDA BIT IN COMMAND
       *                                       (VARIABLE LENGTH)
       *
       * SPECIAL FIELDS FOR PAGING IORES (IN TCBS) ONLY:
       *
00001C                         ORG IOREIALCNT
000018 IOREPGFLG               DS  BL1         PAGING FLAGS
       IOREPGFLGRECL           EQU X'80'       PAGE RECLAIMED DURING THIS
       *                                       PAGEOUT IF SET ON PAGEOUT
       *                                       COMPLETION (DEFERRED USE)
000019                         ORG IOREPGFLG
000018 IOREPGTCB               DS  A           ADDRESS OF ASSOCIATED TCB
00001C IOREPGPAGENO            DS  BL1         PAGE NUMBER FOR REQUEST
00001D IOREPGFLUB              DS  AL3         FLUB ADDRESS FOR REQUEST
       *
       IOREPGEND               EQU *
       IOREPGLENGTH            EQU IOREPGEND-IOREBEGIN
```

```
                OFB
000000 OFB          DSECT
       *
       *        THE OPEN FILE BLOCK (OFB) IS CONSTRUCTED WHEN A FILE IS
       *        OPENED, AND CONTAINS INFORMATION FOR USE BY DATA MANAGEMENT
       *        AND I/O INITIATION ROUTINES WHICH MUST BE PROTECTED FROM
       *        USER PROGRAM MODIFICATION. ALL OPEN FILE BLOCKS ARE RESIDENT
       *        IN THE SYSTEM AREA. ALL OPEN FILE BLOCKS FOR A PARTICULAR
       *        DISK FILE POINT TO THE SAME FILE LOCATION AND USE BLOCK
       *        (FLUB).  WHEN AN OFB IS CREATED, SPACE FOR AN IORE IS
       *        ALLOCATED AS WELL, AND THE IORE ADDRESS IS PLACED IN
       *        OFBIOREPTR.
       *
000000 OFBBEGIN                    DS  0D          (DOUBLEWORD ALIGNMENT REQD)
       * ********************************************************************
       * BASIC SECTION:
       * ********************************************************************
000000 OFBFLAGS                    DS  0BL1        FLAG BYTE
       * MODE FLAGS:
       OFBFLAGSWPSHARE             EQU X'80'       OPENED FOR WP SHARE MODE 1B↑
       OFBFLAGSOUT                 EQU X'40'       OPENED FOR OUTPUT   MODE
       OFBFLAGSIN                  EQU X'20'       OPENED FOR INPUT    MODE
       OFBFLAGSIO                  EQU X'10'       OPENED FOR IO       MODE
       OFBFLAGSEXTEND              EQU X'08'       OPENED FOR EXTEND   MODE
       OFBFLAGSSHARED              EQU X'04'       OPENED FOR SHARED   MODE
       OFBFLAGSCPL2                EQU X'02'       FOR USE BY LEVEL 2 COMMAND
       *                                           PROCESSOR - LEVEL 1 MUST
       *                                           NOT USE
       OFBFLAGSIPCB                EQU X'01'       OPENED VIA "IPCB"
000000 OFBUFB                      DS  A           UFB ADDR (USER FILE BLOCK)
       *                                           (CONTAINS BINARY VALUE -1 IF
       *                                           PRESUPPLIED OFB FOR
       *                                           WORKSTATION USED BY COMMAND
       *                                           PROCESSOR; CONTAINS 0 IF NO
       *                                           UFB'S ARE CHAINED TO THE
       *                                           OFB, OR IF THE OFB WAS
       *                                           OPENED VIA "IPOPEN")
000004 OFBUCB                      DS  A           UCB ADDRESS
       *                                           (ZERO IF DUMMY)
000008 OFBIOREPTR                  DS  A           IORE ADDRESS (FOR XIO SVC)
00000C OFBTCB                      DS  A           ASSOCIATED TCB ADDRESS
000010 OFBSEMA                     DS  D (BL1,AL3,BL1,AL3)
       *                                           I/O COMPLETION SEMAPHORE
000018 OFBXIOCNT                   DS  F           NUMBER OF XIOS TO THIS
       *                                           FILE SINCE OPENED
00001C OFBTASKCHN                  DS  A           NEXT OFB THIS TASK OR ZERO
000020 OFBLINKLEV                  DS  HL1         LINK LEVEL ON WHICH
       *                                           FILE WAS OPENED
000021 OFBFLAG1                    DS  BL1         FLAG1 BYTE                  1↑
       OFBFLAG1LOCKED              EQU X'80'       FILE LOCKED                 1↑
       OFBFLAG1CAN                 EQU X'01'       CLOSE ATTEMPTED BY CANCEL   1↑
       *
```

```
000022                          DS  H           *** SPARE (MUST BE ZERO) 1A↑
       *
       OFBBASICEND              EQU *           END OF BASIC OFB BLOCK

       * ********************************************************************
       * DISK-ONLY SECTION (DEFINED IN OFBS FOR DISK FILES)
       * ********************************************************************
000024                          ORG OFBBASICEND
000024 OFBFLUBPTR               DS  A           ADDR OF FLUB FOR THIS FILE
000028 OFBCFLAGS                DS  0BL1        OFB SPECIAL CLOSE FLAGS
       OFBCFLAGSBYP             EQU X'80'       DENOTES DMS-CLOSE VECTOR
       *                                        SHOULD BE BYPASSED AT CLOSE
       *                                        TIME DUE TO OPEN OR CLOSE
       *                                        ERROR. (ALSO, NO LABEL
       *                                        UPDATE DONE IF SET)
000028 OFBFILECHN               DS  A           CHAIN OF OFBS THIS FILE
       *                                        (HEAD IN FLUB)
00002C OFBBCOUNT                DS  0HL1        COUNT OF BUFFERS IN BUFFER
       *                                        POOL IF PRESENT
00002C OFBBCT                   DS  A           BUFFER POOL CONTROL TABLE
       *                                        ADDRESS, IF ANY
       * ********************************************************************
       * OFB "IPCB" EXTENSION (DEFINED ONLY IF "OFBFLAGSIPCB" SET)
       * ********************************************************************
000030                          ORG OFBBASICEND
000024 OFBIPCB                  DS  A           ADDRESS OF THE "IPCB"
       *                                        CONTROLLING THIS DEVICE
000028                          DS  2A          (RESERVED)
       * ********************************************************************
       OFBEND                   EQU *
       OFBLENGTH                EQU OFBEND-OFBBEGIN
```

```
                TPLAB
000000 TPLAB          DSECT
       *
       *        MAGNETIC TAPE FILE HEADER, TRAILER, AND END OF VOLUME
       *        LABELS CONFORM TO ANSI STANDARDS, AND ARE AS DESCRIBED HERE:
       *        ONLY ID AND BLKCOUNT FIELDS ARE REQUIRED IN EOV1 AND EOF1.
       *
       *        DATE 3/28/79
       *        VERSION 4.00
       *
       TPLABBEGIN                EQU *
000000 TPLABID                   DS  CL4         'HDR1', 'EOV1', OR 'EOF1'
000004 TPLABFILE                 DS  CL17        UP TO 17 ASCII CHARACTERS,
       *                             LEFT ADJUSTED AND PADDED
       *                             WITH BLANKS, NAMING THE FILE
000015 TPLABVOL1SER              DS  CL6         VOLUME SERIAL NUMBER MATCHIN
       *                             'VOL1SER' IN VOLUME LABEL (OF THE
       *                             FIRST VOLUME, IF A MULTI-VOLUME
       *                             FILE)
00001B TPLABFILESECTION          DS  CL4'0001'   ORDER OF VOLUME IN A MULTI-
       *                             VOLUME FILE (ASCII '0001' FOR A
       *                             SINGLE-VOLUME FILE)
00001F TPLABFILESEQ              DS  CL4         FILE SEQUENCE NUMBER
       *                             ON MULTI-FILE VOLUME (1ST FILE
       *                             IS ASCII '0001')
000023 TPLABGENERATION           DS  CL4'0001'   GENERATION NUMBER (CURRENTLY
       *                             ALWAYS '0001', USE DEFERRED)
000027 TPLABVERSION              DS  CL2'00'     VERSION IN GENERATION,
       *                             CURRENTLY ALWAYS ZERO
000029 TPLABCREATION             DS  CL6         CREATION DATE IN THE FORM
       *                             BYYDDD, WHERE B IS A BLANK,
       *                             YY IS YEAR INTO CENTURY,
       *                             DDD IS JULIAN DAY (001 TO 366)
00002F TPLABEXPIRATION           DS  CL6         EXPIRATION DATE IN THE
       *                             ABOVE FORMAT
000035 TPLABACCESS               DS  CL1' '      ACCESS PROTECTION (FILE
       *                             PROTECTION CLASS OR BLANK)
000036 TPLABBLKCOUNT             DS  CL6         BLOCK COUNT IN TRAILER LABEL
       *                             AS SIX ASCII DIGITS. ALWAYS
       *                             PLACED IN 'EOV1' AND 'EOF1'
       *                             LABELS. ASCII ZEROS IN HDR LABEL.
00003C TPLABSYSTEM               DS  CL13        CHARACTERS IDENTIFYING
       *                             THE CREATING SYSTEM
000049 TPLABCREATOR              DS  CL3         FILE CREATOR ID OR BLANKS
00004C TPLABSPARE1               DS  CL4         RESERVED - MUST BE BLANKS
       TPLABEND                  EQU *
       TPLABLENGTH               EQU TPLABEND-TPLABBEGIN
```

```
                TPLB2
000000 TPLB2            DSECT
       *
       *        MAGNETIC TAPE SECONDARY HEADER, TRAILER, AND END OF
       *        VOLUME LABELS CONFORM TO ANSI STANDARDS, AS FOLLOWS:
       *
       *        DATE 3/28/79
       *        VERSION 4.00
       *
       TPLB2BEGIN                   EQU *
000000 TPLB2ID                      DS  CL4 'HDR2', 'EOV2',          OR 'EOF2'
000004 TPLB2RECFM                   DS  CL1 'F' - FIXED LENGTH RECORDS
       *
       *                                    'F' - FIXED LENGTH RECORDS
       *                                    'D' - VARIABLE LENGTH RECORDS
       *                                          IBM FORMAT
       *                                    'W' - VARIABLE LENGTH RECORDS
       *                                          WANG FORMAT
       *                                    'X' - VARIABLE LENGTH COMPRESSED
       *                                          RECORDS WANG FORMAT
       *                                    'U' - UNDEFINED LENGTH RECORDS
000005 TPLB2BLKL                    DS  CL5 BLOCK LENGTH (ASCII)
00000A TPLB2RECL                    DS  CL5 RECORD LENGTH (ASCII)
00000F TPLB2ORG                     DS  BL1 FILE ORGANIZATION
       TPLB2ORGCONSEC               EQU X'01'       CONSECUTIVE
       TPLB2ORGPRINT                EQU X'40'       PRINT FILE
       TPLB2ORGPROG                 EQU X'80'       PROGRAM FILE
000010 TPLB2SPARE1                  DS  CL34        RESERVED FOR OPERATING
       *                                SYSTEM USE
000032 TPLB2BOFF                    DS  CL2 BUFFER OFFSET
000034 TPLB2SPARE2                  DS  CL28        RESERVED - MUST BE ASCII BLA
       TPLB2END                     EQU *
       TPLB2LENGTH                  EQU TPLB2END-TPLB2BEGIN
```

```
                UFB
000000 UFB          DSECT
       *
       *        THE USER FILE BLOCK (UFB) IS SUPPLIED IN THE USER'S
       *        MODIFIABLE AREA BY THE USER'S PROGRAM BEFORE OPENING
       *        A FILE, AND IS ADDRESSED TO REQUEST EACH OPERATION
       *        ON THAT FILE.  THE ADDRESS OF THIS BLOCK IS PLACED
       *        IN THE OPEN FILE BLOCK BY 'OPEN', AND THE ADDRESS OF
       *        THE OPEN FILE BLOCK IS PLACED IN THIS BLOCK.
       *
       *        DATE  1-9-81
       *        VERSION 5.4
       *
000000 UFBBEGIN                  DS  0F          (FULLWORD ALIGNMENT REQUIRED
       * ********************************************************************
       * ACCESS METHOD SECTION
       * NO FIELDS NEED BE SUPPLIED BEFORE 'OPEN', BUT UFBERRAD
       * UFBEODAD, UFBRECAREA, AND UFBKEYAREA MAY BE PRESET
       * IF DESIRED. AFTER 'OPEN', THE USER'S PROGRAM NORMALLY
       * HAS OCCASION TO MODIFY ONLY THIS SECTION OF THE UFB.
       * THE FIRST BYTES OF EACH OF UFBVREAD, UFBVWRITE, UFBVREWRITE,
       * UFBVDELETE AND UFBVSTART ARE ZEROED BY 'OPEN' AND SET
       * THEREAFTER TO FUNCTION MODIFIER VALUES BY THE USER'S PROGRAM.
       * THE SUCCEEDING BYTES OF THESE FIELDS CONTAIN ADDRESSES
       * SUPPLIED BY 'OPEN' WHICH SHOULD NOT BE ALTERED BY THE
       * USER'S PROGRAM WHILE THE FILE IS OPEN.
       * UFBFS1 AND UFBFS2 ARE SET TO X'30' BY 'OPEN' AND MODIFIED
       * THEREAFTER BY DATA MANAGEMENT FUNCTIONS.
       * ********************************************************************
000000 UFBVECT                   DS  5A          BRANCH POINTS TO ACCESS
       *                                        METHOD ROUTINES
       * *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
       * THE FOLLOWING FUNCTION MODIFIER VALUES ARE PLACED IN THE FIRST
       * BYTE OF THE WORD CONTAINING THE ADDRESS OF THE FUNCTION TO BE
       * PERFORMED FOR A USER PROGRAM BEFORE BRANCHING TO THE ROUTINE
       * ADDRESS.
000014                      ORG UFBVECT
000000 UFBV                      DS  0F          (PREFIX TO EQUATE LABELS)
       * MODIFIERS FOR READ:
       UFBVHOLD                  EQU X'01'       (HOLD BLOCK EXCLUSIVELY)
       UFBVREL                   EQU X'04'       (RELATIVE READ)
       UFBVKEYED                 EQU X'04'       (KEYED READ)
       UFBVNODATA                EQU X'08'       (DO NOT MOVE DATA TO WORK
       *                                         AREA ON READ)
       * MODIFIERS FOR READ OR REWRITE (WORKSTATION ONLY):
       UFBVTABS                  EQU X'10'       (READ OR REWRITE TABS - WS)
       * MODIFIERS FOR READ (WORKSTATION ONLY):
       UFBVMOD                   EQU X'02'       (READ MODIFIABLE - WS)
       UFBVALTR                  EQU X'40'       (READ ALTERED - WS)
       * MODIFIERS FOR REWRITE (WORDSTATION ONLY):

       UFBVSELW                  EQU X'40'       (REWRITE SELECTED - WS)
```

```
        * MODIFIERS FOR START (INPUT, IO, SHARED MODES; INDEXED DISK ONLY):
        UFBVEQ                  EQU X'01'       (EQUAL TO)
        UFBVGT                  EQU X'02'       (GREATER THAN)
        UFBVGE                  EQU X'03'       (GREATER THAN OR EQUAL TO)
        * MODIFIER FOR START (SHARED MODE; IGNORED FOR INPUT & IO MODES):
        UFBVHFILE               EQU X'80'       (HOLD FILE)
        UFBVRLS                 EQU X'20'       (RELEASE HELD FILE)
        UFBVRANGE               EQU X'04'       (HOLD REQUEST FOR A RANGE)
        UFBVRETRIEVAL           EQU X'40'       (HOLD CLASS IS RETRIEVAL)
        UFBVLIST                EQU X'10'       (LIST OPTION)
        * MODIFIERS FOR START (CONSECUTIVE OUTPUT & EXTEND MODES ONLY):
        UFBVINPUT               EQU X'04'       (CHANGE TO TEMPORARY IO MODE
        UFBVOUTPUT              EQU X'08'       (CHANGE TO OUTPUT MODE)
        UFBVEXTEND              EQU X'20'       (CHANGE TO EXTEND MODE)
        * MODIFIERS FOR START (CONSECUTIVE FILES WITH VARIABLE-LENGTH
        * RECORDS, INPUT AND I/O MODES ONLY):
        UFBVBEGIN               EQU X'10'       (BEGINNING OF FILE)
        UFBVSKIP                EQU X'40'       (FROM CURRENT RECORD
        *                                        USING SIGNED WORD
        *                                        ADDRESSED BY KEYAREA)
        * MODIFIERS FOR START (PHYSICAL ACCESS METHOD ONLY):
        UFBVCMD                 EQU X'80'       (***VAGUE NOTE***)
        UFBVWAIT                EQU X'40'       (WAIT FOR I/O COMPLETION)
        UFBVWAITS               EQU X'41'       WAIT FOR TC I/O COMPLETION
        *                                       ON THIS DEVICE ONLY
        UFBVWAITM               EQU X'42'       WAIT FOR TC I/O COMPLETION
        *                                       ON ALL DEVICES OPENED BY
        *                                       THIS PROGRAM
        UFBVWAITA               EQU X'43'       WAIT FOR TC I/O COMPLETIONS
        *                                       AND TC UNSOLICIT INTERRUPTS
        UFBVHALTIO              EQU X'20'       HALT TC IO OPERATION
        * MODIFIERS FOR START (WORKSTATION ONLY):
        UFBVATINT               EQU X'10'       (TEST FOR ATTENTIONS RECEIVE
        * *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
000000                  ORG UFBVECT
000000 UFBVREAD                 DS  A           ..FOR READ
000004 UFBVWRITE                DS  A           ..FOR WRITE
000008 UFBVREWRITE              DS  A           ..FOR REWRITE
00000C UFBVDELETE               DS  A           ..FOR DELETE
000010 UFBVSTART                DS  A           ..FOR START
        * THE FOLLOWING FOUR FIELDS MAY BE SET BEFORE 'OPEN' OR
        * BEFORE THE FIRST FUNCTION AFTER 'OPEN'. THEY MAY BE CHANGED
        * BY THE USER'S PROGRAM BEFORE ANY FUNCTION. IF UFBEODAD IS 0,
        * UFBERRAD WILL BE USED FOR END OF DATA AND INVALID-KEY CONDITIONS.
        * IF UFBERRAD IS 0, ABNORMAL TERMINATION WILL OCCUR ON ANY
        * ERROR (AND ON THE ABOVE CONDITIONS IF UFBEODAD IS 0  ALSO).
000014 UFBERRAD                 DS  A           I!O UNUSUAL CONDITION USER
        *                                   ROUTINE ENTRY POINT, OR ZERO
000018 UFBEODAD                 DS  A           END OF DATA AND INVALID KEY
        *                                   USER ROUTINE
        *                                   ENTRY POINT, OR ZERO.
```

```
00001C UFBRECAREA              DS  A          ADDRESS IN USER-MODIFIABLE S
       *                                  OF RECORD WORK AREA
000020 UFBKEYAREA              DS  A          ADDRESS OF AREA CONTAINING
       *                                  SUPPLIED KEY OR RECORD NUMBER
       *                                  FOR START OR READ FUNCTIONS
       *                                  (IF ZERO FOR WORKSTATION FILES,
       *                                  LINE NUMBER (ROW) TAKEN FROM ORDER
       *                                  AREA)
000024 UFBFS1                  DS  CL1        FILE STATUS BYTE 1 FOR DMS
       UFBFS1SUCCESS           EQU X'30'      SUCCESSFUL COMPLETION
       UFBFS1ATEND             EQU X'31'      AT END
       UFBFS1INVKEY            EQU X'32'      INVALID KEY OR RECORD NO.
       UFBFS1IOERR             EQU X'33'      PERMANENT I/O ERROR
       UFBFS1ADMSERR           EQU X'34'      ADMS FUNCTION ERROR
       UFBFS1CANCEL            EQU X'36'      CANCEL CODE STORED
       *                       FOR UFBF1NOMSG (OPEN,DMS,CLOSE); UFBFS2=C'0'
       *                       MSGID AT UFBVREAD FOR O/C; NO MSGID IF DMS
       UFBFS1TIME              EQU X'37'      TIME-OUT CONDITION ON
       *                                      SHARED MODE RESOURCE WAIT
       UFBFS1SHARE             EQU X'38'      FS FOR SHARER CONDITION
       *                                      RESOURCE WAIT
       UFBFS1OTHER             EQU X'39'      OTHER CONDITIONS
       **
000025 UFBFS2                  DS  CL1        FILE STATUS BYTE 2 FOR DMS
       UFBFS2NOINFO            EQU X'30'      NO FURTHER INFO
       **
       * THE FOLLOWING UFBFS2 VALUES ARE SET WITH UFBFS1INVKEY (X'32')
       UFBFS2SEQERR            EQU X'31'      SEQUENCE ERROR
       UFBFS2DUPKEY            EQU X'32'      DUPLICATE KEY
       UFBFS2NOREC             EQU X'33'      NO RECORD FOUND
       UFBFS2BYVIOL            EQU X'34'      BOUNDARY VIOLATION
       * UFBFS2BDYVIOL IS ALSO USED WITH UFBFS1IOERR (FS = C'34')
       **
       * THE FOLLOWING UFBFS2 VALUES ARE SET WITH UFBFS1ADMSERR (X'34')
       UFBFS2INTS              EQU X'31'      ATTEMPT TO UPDATE FILE
       *                                      WHILE NOT IN A TRANSACTION
       UFBFS2MCC               EQU X'32'      MCC ERROR
       UFBFS2ICK               EQU X'33'      ICK VIOLATION
       UFBFS2LOG               EQU X'34'      UNABLE TO LOG RECORD IMAGE
       UFBFS2VAB               EQU X'35'      INVALID VAB INFORMATION
       UFBFS2CRASH             EQU X'36'      FILE PREVIOUSLY CRASHED
       UFBFS2DATA              EQU X'37'      ICK CHECK FAILED
       UFBFS2UNQERR            EQU X'38'      ADMS FILE WITH RECOVERY, 04↑
       *   HAS ALTERNATE INDEX WITH NO DUPS; USER MUST HOLD ENTIRE FILE TO 04↑
       *   MAKE UPDATE, BUT DIDN'T HOLD WHOLE FILE.                        04↑
       *
       * THE FOLLOWING UFBFS2 VALUES ARE SET WITH UFBS1SHARE (X'38')
       UFBFS2ACC               EQU X'35'      UPDATE ACCESS DENIED FOR
       *                                      USER WITH READ-ONLY RIGHTS
       *                                      IN SHARED MODE
       UFBFS2RESERR            EQU X'36'      RESOURCE CONTROL ERROR
       **
       * THE FOLLOWING UFBFS2 VALUES ARE SET WITH UFBFS1OTHER (X'39')
       UFBFS2INVFUN            EQU X'35'      INVALID FUNCTION OR
       *                                      FUNCTION SEQUENCE
```

```
UFBFS2INVCMD              EQU X'36'       INVALID COMMAND (ALIGNMENT
*                                       OR ADDRESS ERROR FOR DIRECT I/O)
UFBFS2INVLTH              EQU X'37'       INVALID LENGTH
UFBFS2MASK                EQU X'38'       INVALID ACCESS MASK
*                                         (ALTERNATE INDEXED FILES)
UFBFS2TRLERR              EQU X'38'       TRAILER COUNT NOT EQUAL
*                                       TO BLOCKS READ (SET BY SVC
*                                       CLOSE ONLY)
UFBFS2FMTERR              EQU X'39'        FORMAT ERROR (BLOCK PREFIX,
*                                       RECORD PREFIX,EXPANSION ERROR OR
*                                       INVALID CHAIN FIELD)
**
* NOTE: UFBFS2 CONTAINS THE TERMINATING ATTENTION CHARACTER (AID BYTE)
*       ON WORKSTATION READ SUCCESSFUL COMPLETION.
**
* NOTE: THE FOLLOWING UFBFS2 VALUES ARE SET ONLY IF AN SVC OPEN
*       EXIT IS TAKEN. THESE VALUES ARE ALSO USED WHEN CREATING
*       THE OPEN EXIT MASK TO BE SUPPLIED TO THE OPEN SVC.
UFBFS2XFILE               EQU X'80'       DUPLICATE FILE OR
*                                         FILE NOT FOUND
UFBFS2XLIB                EQU X'40'       LIBRARY NOT FOUND
UFBFS2XVOL                EQU X'20'       VOLUME NOT MOUNTED
UFBFS2XSPACE              EQU X'10'       NO SPACE ON VOLUME
UFBFS2XVTOC               EQU X'08'       NO VTOC SPACE ON VOLUME
UFBFS2XTAPELD             EQU X'08'       WRONG TAPE LABEL/DENSITY
UFBFS2XPOS                EQU X'04'       POSSESSION CONFLICT
UFBFS2XPROT               EQU X'02'       PROTECTION CLASS VIOLATION
UFBFS2XFORMAT             EQU X'01'       OPEN FORMAT ERROR - ERROR
*                                         CLASS DESCRIBED IN UFBXCODE
UFBAMEND                  EQU *
UFBAMLENGTH               EQU (UFBAMEND-UFBBEGIN)
* **********************************************************************
* FILE LOCATION AND ATTRIBUTE SECTION
* ALL FIELDS IN THIS SECTION MUST BE SET (SOME OF THEM POSSIBLY
* TO 'NULL' VALUES) BY THE USER'S PROGRAM BEFORE INITIALLY
* ADDRESSING AN 'OPEN' TO THE UFB.
* ALL RELEVANT FIELDS AND FLAGS SET NULL BEFORE 'OPEN' ARE SUPPLIED
* HERE BY 'OPEN' PROCESSING AND MAY BE EXAMINED BY THE USER'S
* PROGRAM. THE PROGRAM SHOULD NOT MODIFY THESE FIELDS BETWEEN
* 'CLOSE' AND A SUCCESSIVE 'OPEN' IF THE SAME FILE IS REQUIRED
* (WITHOUT REPROMPTING).
* **********************************************************************
000026 UFBBLKSIZE              DS  H   MAGNETIC TAPE - MUST CONTAIN
*                                     PHYSICAL BLOCK SIZE BEFORE OPEN
*                                     IF OUTPUT MODE OR UNLABELLED
*                                     TAPE.
*                                     DISK OR DISKETTE - ALWAYS 2048
*                                     AFTER OPEN EXCEPT WHEN USING
*                                     PHYSICAL ACCESS METHOD (PAM)
000028 UFBRECSIZE              DS  H          LOGICAL RECORD SIZE
*                                     (MUST BE SUPPLIED BEFORE OPEN FOR
*                                     OUTPUT OPEN MODE)
```

```
00002A UFBFORG                 DS  BL1         FILE ORGANIZATION
       UFBFORGCONSEC           EQU X'01'       CONSECUTIVE
       UFBFORGINDEXED          EQU X'02'       INDEXED
       UFBFORGWP               EQU X'04'       WORD PROCESSING FILE
       UFBFORGVIBM             EQU X'08'       IBM VARIABLE-LENGTH RECORDS
       UFBFORGU                EQU X'10'       UNDEFINED RECORD FORMAT
       UFBFORGVLEN             EQU X'20'       VARIABLE-LENGTH RECORDS
       UFBFORGPRINT            EQU X'40'       PRINT FILE
       UFBFORGPROG             EQU X'80'       PROGRAM FILE
       *
00002B UFBF1                   DS  BL1         OPTION FLAGS
       UFBF1NOGET              EQU X'80'       USE GETPARM = TYPE RD
       UFBF1NODISP             EQU X'40'       USE GETPARM = TYPE ID
       * UFBF1NOGET AMD UFBF1NODISP USED BY SVC OPEN ONLY; NOT RESET BY DMS
       UFBF1PAM                EQU X'20'       PHYSICAL ACCESS METHOD
       UFBF1BAM                EQU X'10'       BLOCK ACCESS METHOD
       UFBF1PREVO              EQU X'08'       THIS UFB PREVIOUSLY OPENED
       UFBF1WORK               EQU X'04'       SCRATCH THIS WORK FILE ON
       *                                       CLOSE IF SET & FILE HAS A
       *                                       TEMPORARY NAME
       UFBF1POOL               EQU X'02'       BUFFER POOLING FOR RAM
       *                                       (UFBBUFSTART MUST CONTAIN
       *                                       BCT ADDRESS AT OPEN TIME)
       UFBF1OPEN               EQU X'01'       THIS UFB OPEN IF SET
00002C UFBF2                   DS  BL1         OPEN MODE FLAGS
       UFBF2ADMS               EQU X'80'       TO OPEN IN ADMS MODE
       UFBF2OUT                EQU X'40'       TO OPEN FOR OUTPUT MODE
       UFBF2IN                 EQU X'20'       TO OPEN FOR INPUT MODE
       UFBF2IO                 EQU X'10'       TO OPEN FOR IO MODE
       UFBF2EXTEND             EQU X'08'       TO OPEN FOR EXTEND MODE
       UFBF2SHARED             EQU X'04'       TO OPEN FOR SHARED MODE
       UFBF2DALT               EQU X'02'       DELETIONS IN PROGRESS    00↑
       *                                        ON ALT-INDEX FILE       00↑
00002D UFBDEVCLASS             DS  BL1         DEVICE CLASS (REQUIRED
       *                                       BY 'OPEN')
       UFBDEVCLASSWS           EQU X'01'       WORKSTATION
       UFBDEVCLASSTAPE         EQU X'02'       MAGNETIC TAPE
       UFBDEVCLASSDISK         EQU X'03'       DISK
       UFBDEVCLASSPRT          EQU X'04'       PRINTER
       UFBDEVCLASSTC           EQU X'05'       TC DEVICE
       UFBDEVCLASSDUMM         EQU X'FF'       DUMMY FILE
00002E UFBFLAGS                DS  BL1         FILE ATTRIBUTE FLAGS
       UFBFLAGSUPDAT           EQU X'80'       FILE HAS BEEN CLOSED
       UFBFLAGSCOMP            EQU X'40'       DATA RECORDS IN COMPRESSED
       *                                       FORMAT
       * ******* UFBFLAGSRECOV - RECOVERY=YES FOR BIT = ZERO **********
       UFBFLAGSRECOV           EQU X'20'       USE PREFORMAT AND RECOVERY
       *                                  PROCEDURES IF ZERO (INDEXED ONLY)
       UFBFLAGSALTX            EQU X'10'       ALTERNATE INDICES IN FILE
       UFBFLAGSLOG             EQU X'08'       CONSEC LOG FILE FLAG
       UFBFLAGSALTP            EQU X'08'       ALTERNATE-TREE PROCESS FLAG
```

```
        UFBFLAGSPART    EQU X'04'       PARTIAL BACKUP FILE
        *                       PROGRAM SETS BIT BEFORE OPEN OUTPUT
        *                       (BAM OR PAM) TO SET BIT IN FILE
        *                       LABEL, OR SETS BIT BEFORE NON-OUTPUT
        *                       OPEN (BAM OR PAM) IF ABLE TO PROCESS
        *                       PARTIAL FILES. INVALID FOR RAM.
        UFBFLAGSADMS    EQU X'02'       ADMS DISK FILE INDICATOR
        UFBFLAGSPRIV    EQU X'01'       PROGRAM FILE CARRIES
        *                               ADDITIONAL ACCESS PRIVILIGES
00002F  UFBDEVADDR      DS  HL1         DEVICE ADDRESS (FOR PRINTERS
        *                       AND WORKSTATIONS ONLY.
        *                       USED IF SUPPLIED
        *                       AND PLACED HERE BY 'OPEN' IF
        *                       NOT SUPPLIED. HEX FF IF
        *                       NOT SUPPLIED.)
000030  UFBF3           DS  0BL1 (* NAME KEPT FOR COMPATIBILITY *)
000030  UFBPRTCLASS     DS  CL1         PRINT CLASS (A-Z)
        *
000031  UFBFORMNO       DS  HL1         PRINTER FORM NUMBER (BINARY)
000032  UFBPRNAME       DS  CL8         PARAMETER REFERENCE NAME
        *                       (MUST ALWAYS BE SUPPLIED HERE
        *                       FOR 'OPEN')
00003A  UFBVOLSER       DS  CL6         VOLUME SERIAL NUMBER FOR
        *                       VOLUME-ORIENTED FILES (TAPE
        *                       OR DISK)
        *                       (IF 6 ASCII BLANKS, TAKEN FROM
        *                       PROCEDURE SPECIFICATION OR
        *                       'OPEN'-TIME PROMPT. IF SPECIFIED
        *                       IN NEITHER OF THESE WAYS,
        *                       TAKEN FROM DEFAULT IN
        *                       ETCB)
000040  UFBDIRNAME      DS  CL8         DIRECTORY NAME (IF 8 ASCII
        *                       BLANKS, DIRECTORY NAME TAKEN
        *                       FROM PROCEDURE SPECIFICATION
        *                       OR 'OPEN'-TIME PROMPT.
        *                       IF SPECIFIED IN NEITHER PLACE
        *                       AND VOLUME SERIAL ALSO
        *                       OMITTED, DEFAULT IN ETCB
        *                       USED)

000048  UFBFILENAME     DS  CL8         FILE NAME (UNDER DIRECTORY)
        *                       (IF 8 BLANKS, FILE NAME TAKEN
        *                       FROM PROCEDURE SPECIFICATION
        *                       OR 'OPEN'-TIME PROMPT.
        *                       WORK FILE SPECIFICATION IF
        *                       ASCII '#' OR '$' FOLLOWED BY
        *                       FOUR ALPHAMERICS - LAST
        *                       3 CHARACTERS THEN MUST BE
        *                       BLANKS - SEE WORK FILE
        *                       DOCUMENTATION)
```

```
000050 UFBFPCLASS              DS  CL1         FILE PROTECTION CLASS
       *                           VALUE TO LABEL IF OUT-MODE;
       *                           TAKEN FROM USER 'SET' DEFAULTS IF
       *                           X'00' IS SUPPLIED;
       *                           VALUE FROM LABEL IF EXISTING FILE
000051 UFBCREATOR              DS  CL3         FILE CREATOR FOR NEW OR
       *                           EXISTING DISK FILES
000054 UFBALTCNT               DS  OBL1        COUNT OF ALTERNATE INDICES
       *                           IN FILE AFTER SVC OPEN
000054 UFBALTPTR               DS  A           POINTER TO AXD1-AREA FOR DMS
       *                           PROCESSING (ALL REFERENCE TO THE
       *                           AXD1-AREA MUST USE UFBALTPTR)
       *
       * FOR DEVICES OTHER THAN DISK, THE ALTCNT FIELD IS FOR MICROCODE TYPE
000058                         ORG UFBALTCNT
000054 UFBMCTYPE               DS  XL1         DEVICE TYPE
       UFBMCTYPE2780           EQU X'01'       2780 BATCH TC
       UFBMCTYPE3780           EQU X'02'       3780 BATCH TC
       UFBMCTYPETCD            EQU X'03'       TC DIAGNOSTICS
       *
       * FOR TC2780, TC3780 FILES, THE ALTPTR FIELD IS USED FOR THE TC
       * BATCH STREAM OPTIONS
000055 UFBTCDATAOPT            DS  BL1         TC STREAM DATA OPTION
000056 UFBTCXMITOPT            DS  BL1         TC STREAM TRANSMIT/RECEIVE
       *                                       OPTION
000057 UFBTCMAXRECSZ           DS  XL1         TC STREAM MAXIMUM RECSIZE
       *                                       MINUS 1
       * FOR WORD PROCESSING WORKSTATIONS, THE ALTPTR FIELD IS USED FOR
       * EXTENDED WS-ATTENTION INFORMATION
000058                         ORG UFBALTPTR+1
000055 UFBWPAID                DS  XL3         EXTEND WS-ATTN INFORMATION
       **
000058 UFBF4                   DS  BL1         ADDITIONAL DEVICE-DEPENDENT
       *                                       FLAGS
       UFBF4NOVTOC             EQU X'80'       UNSTRUCTURED DISKETTE
       UFBF4RLSE               EQU X'40'       RELEASE UNUSED SPACE
       *                                       ON CLOSE
       UFBF4BLKAL              EQU X'20'       ALLOCATE SPACE FOR NEW
       *                                       DISK FILE IN BLOCKS,
       *                                       FROM UFBNBLKS

       UFBF4VERIFY             EQU X'10'       VERIFY OPTION ON ALL
       *                                       DISK WRITES
       UFBF4NOMSG              EQU X'08'       NO RESPECIFY OR CANCEL
       *                                       MESSAGE FOR SVC OPEN
       *                                       ALSO NO CANCEL ON CLOSE; NO
       *                                       ACK/CANCEL FOR DMS.
       UFBF4NOACK              EQU X'04'       NO EXCEPTIONAL CONDITION
       *                                       ACKNOWLEDGMENT MESSAGES
       *                                       FOR DMS FUNCTIONS
       UFBF4PMSG               EQU X'02'       FOR INTERNAL USE BY DMS -
       *                                       CLOSE SENDS MESSAGE TO
       *                                       UNSPOOLER IF SET
```

```
        UFBF4ALLOWT             EQU X'01'       USED BY SVC OPEN. PROGRAM
        *                                       SUPPLIES BIT=1 TO ALLOW DEV=TAPE.
        *                                       (OPEN SETS=1 IF UFBDEV=TAPE ALSO)
        *                                       OTHERWISE, DEV=TAPE NOT ACCEPTED.
000059  UFBNRECS                DS  FL3         NUMBER OF DATA RECORDS IN
        *                                       FILE (EXAMINED BY 'OPEN' FOR
        *                                       OUTPUT OPEN MODE ONLY.
        *                                       EXCLUDES INDEX RECORDS, ETC)
00005C  UFBLRECSAVE             DS  H           RECSIZE SAVED HERE
        *                                       BY OPEN (BAM)
00005E  UFBRETPD                DS  H           RETENTION PERIOD IN DAYS
        *                                       (MAXIMUM 999)
        UFBLOCEND               EQU *
        UFBLOCLENGTH            EQU (UFBLOCEND-UFBBEGIN)
        * ***************************************************************
        * DATA MANAGEMENT SYSTEM SECTION
        * ***************************************************************
000060  UFBBCB1                 DS  BL16        BUFFER CONTROL BLOCK
        *                                       (CORRESPONDS TO SVC XIO PARAMETER
        *                                       LIST)
000070                          ORG UFBBCB1
000060  UFBXIOFLAGS             DS  0BL1        FLAG BYTE FOR SVC XIO
        UFBXIOFLAGSRLS          EQU X'80'       RELEASE BUFFER AFTER WRITE
000060  UFBOFB                  DS  A           OFB ADDRESS
000064  UFBBUFCMD               DS  0BL1        COMMAND BYTE FOR OPERATION
000064  UFBBUFADR               DS  A           BUFFER MEMORY ADDRESS
        *                                       (BLOCK ADDRESS WITHIN
        *                                       BUFFER IF BUFFER LARGER
        *                                       THAN 2K)
000068  UFBBUFDATAL             DS  H           LENGTH IN BYTES FOR
        *                                          OPERATION
00006A  UFBBUFOFFSET            DS  H           OFFSET OF NEXT RECORD
        *                                          IN BUFFER
00006C  UFBBUFBLOCK             DS  FL3         (STARTING) BLOCK WITHIN
        *                                          FILE OF BUFFERED DATA
00006F  UFBBCBFLAGS             DS  BL1         FLAGS
        UFBBCBFLAGSLOD          EQU X'01'       BUFFER CONTENTS VALID
        UFBBCBFLAGSTOR          EQU X'02'       BUFFER TO BE REWRITTEN
        UFBBCBFLAGSIO           EQU X'04'       BUFFER I/O IN PROGRESS
        UFBBCBFLAGSPROT         EQU X'10'       BUFFER IN PROTECTED MEMORY
        UFBBCBFLAGSEOB          EQU X'20'       END OF BLOCK REACHED
        UFBBCBFLAGSEOF          EQU X'40'       EOF BLOCK IN BUFFER
        **
        * THE FOLLOWING FIELDS ARE USED FOR THE TIME-OUT OPTION IN SHARED
        * MODE ONLY.
000070                          ORG UFBBUFDATAL
000068  UFBTIMEEXIT             DS  A           EXIT ADDRESS FOR TIME-OUT
        *                                       RETURN (0 = NO TIME-OUT)
00006C  UFBHOLDID               DS  CL3         INITIALS OF HOLDER OF
        *                                       RESOURCE
00006F  UFBTIME                 DS  XL1         WAIT TIME IN SECOND
        *                                       (0 = NO WAIT)
        **
```

```
000070 UFBBUFSIZE                DS  H           BUFFER SIZE
000072 UFBCHKSIZE                DS  H           RESIDUAL COUNT FROM XIO
       *                                         (DMS USE ONLY)
       * UFBXDATE OR UFBOUTRECS IS AVAILBLE AFTER SVC OPEN AND BEFORE THE
       * FIRST DMS REQUEST; UFBRES3 IS AN INTERNAL DMS FIELD AFTERWARDS.
000074 UFBRES3                   DS  BL3         RESERVED FOR INTERNAL DMS
000077          ORG   UFBRES3
000074 UFBXDATE                  DS  BL3         EXPIRATION DATE (EXIST FILE)
000077          ORG   UFBRES3
000074 UFBOUTRECS                DS  FL3         NUMBER OF RECORDS REQUESTED
       *                                         FOR OUTPUT MODE
000077 UFBNBLKS                  DS  FL3         NUMBER OF 2048-BYTE BLOCKS
       *                                         IN THE FILE
00007A          ORG   UFBNBLKS
000077 UFBDMSGID                 DS  BL3        STORED MSG-ID(DMS NOMSG EXIT)
00007A UFBMAXTFR                 DS  H           MAXIMUM DATA TRANSFER IN
       *                                         BYTES FOR DISK (SET BY OPEN)
00007C          ORG   UFBMAXTFR
00007A UFBRES1                   DS  BL1         FUTURE SPARE BYTE
00007B UFBOPFLAGS                DS  BL1         INTERNAL OPEN FLAGS
       UFBOPFLAGSPFA             EQU X'80'      PRINT-FILE ASSIGNMENT TO DISK
       UFBOPFLAGSPFS             EQU X'40'      PF - USER SUPPLIED FILE NAME
       UFBOPFLAGSWKA             EQU X'20'      WORK-FILE ASSIGNMENT BY OPEN
       UFBOPFLAGSPVS             EQU X'10'      PF - USER SUPPLIED VOLUME
       UFBOPFLAGSSCAN            EQU X'08'      IN SCAN BIT (WORK/SPOOL)
       **
00007C UFBLF                     DS  BL1         LAST FUNCTION PERFORMED
       UFBLFOPEN                 EQU X'00'       OPEN
       UFBLFREAD                 EQU X'04'       READ
       UFBLFWRITE                EQU X'08'       WRITE
       UFBLFREWRITE              EQU X'0C'       REWRITE
       UFBLFDELETE               EQU X'10'       DELETE
       UFBLFSTART                EQU X'14'       START
       UFBLFCLOSE                EQU X'18'       CLOSE
00007D UFBLFMOD                  DS  BL1         LAST FUNCTION MODIFIER
       *                                 (DOESN'T CHANGE ON 'REWRITE')
       *                                 (SEE UFBV ABOVE)
00007E          ORG   UFBLFMOD
00007D UFBXCODE                  DS  BL1         EXTENDED OPEN EXIT CODE
       * UFBXCODE VALUES 1-8 SET FOR POSSESSION CONFLICT
       UFBXCODENOINFO            EQU X'00'       NO FURTHER INFORMATION
       UFBXCODEUSE               EQU X'01'       DEVICE IN USE
       UFBXCODEDET               EQU X'02'       DEVICE DETACHED
       UFBXCODEVOLX              EQU X'03'       VOLUME EXCLUSIVE
       UFBXCODEPOSS              EQU X'04'       FILE POSSESSION CONFLICT
       UFBXCODEPAGE              EQU X'05'       PAGING FILE - SYSTEM ONLY
       UFBXCODEIMAG              EQU X'06'       IMAGE FILE (INPUT MODE ONLY)
       UFBXCODEAOPEN             EQU X'07'       ALREADY OPEN - THIS USER
       UFBXCODEAUSE              EQU X'08'       ALREADY IN USE - THIS USER
       *
       * UFBXCODE VALUES X'10' - X'1F' SET FOR OPEN FORMAT ERROR
       UFBXCODETRACK             EQU X'10'       PROGRAM REQUIRES 7 TRACK
       *                                         TAPE WHILE DRIVE IS 9 TRACK
       *                                         OR VICE VERSA
```

```
        UFBXCODEDNPRT           EQU X'11'           UFB FORG=PRINT, WHILE     LCK
        *                                           FDR FORG NOT= PRINT       LCK
        UFBXCODEDNPRG           EQU X'12'           UFB FORG=PROG, WHILE      LCK
        *                                           FDR FORG NOT= PROG        LCK
        UFBXCODEDNCSC           EQU X'13'           UFB FORG=CONSEC, WHILE    LCK
        *                                           FDR FORG NOT= CONSEC      LCK
        UFBXCODEDNWP            EQU X'14'           UFB FORG=WP,  WHILE       LCK
        *                                           FDR FORG NOT= WP          LCK
        UFBXCODEDNINX           EQU X'15'           UFB FORG=INDEXED, WHILE   LCK
        *                                           FDR FORG NOT= INDEXED     LCK
        UFBXCODEDFGR            EQU X'16'           UFB FORG NEITHER CONSEC   LCK
        *                                           NOR INDEXED---ERROR       LCK
00007E  UFBEREC                    DS  H            LAST RECORD NUMBER WITHIN
        *                                   LAST BLOCK
000080  UFBVERSION                 DS  HL1          UFB VERSION NUMBER *******
000081  UFBEBLK                    DS  FL3          LAST BLOCK NO. WITHIN FILE
        *                                   FROM 0
000084  UFBBUFSTART                DS  A            BUFFER MEMORY ADDRESS;
        *                                           BUFFER CONTROL TABLE
        *                                           ADDRESS BEFORE 'OPEN'
        *                                           IF BUFFER POOLING
        *                                           SPECIFIED (UFBF1POOL SET)
000088  UFBRDLTH                   DS  H            LENGTH IN BYTES OF
        *                                           DATA IN BUFFER
00008A  UFBPRTCOPIES               DS  H            NUMBER OF PRINT COPIES
        *                                           (FOR PRINTER FILES ONLY)
00008C                             ORG UFBPRTCOPIES
00008A  UFBWPBLKSIZE               DS  X            WORD PROCESSING FILE CONTROL
00008B  UFBWPBLS                   DS  X            FIELDS, WP FILES BLKSIZE
        *                                           AND BYTES IN LAST SECTOR
00008C                             ORG UFBBUFSTART
000084  UFBPTRB                    DS  FL4          FIRST BLOCK IN INDEX
        *                                   AREA OF PRIMARY EXTENT
        *                                   (INDEXED FILES)
000088  UFBPTRC                    DS  FL4          LAST BLOCK IN INDEX AREA
        *                                   OF PRIMARY EXTENT
        *                                   (INDEXED FILES)
        UFBDMSEND                  EQU *
        UFBDMSLENGTH               EQU (UFBDMSEND-UFBBEGIN)
        * **********************************************************************
        * END OF UFB FOR ALL FILES/DEVICES EXCEPT TAPE FILES, INDEXED DISK
        * FILES AND ADMS(DATA AND RESTART) DISK FILES.


        * **********************************************************************
        * INDEXED DISK FILE EXTENSION SECTION:
        * UFBKEYPOS AND UFBKEYSIZE SHOULD BE FILLED IN BY THE PROGRAM BEFORE
        * 'OPEN' FOR A NEW INDEXED FILE (UFBF2OUT AND UFBFORGINDEXED  SET).
        * THEY ARE SET BY 'OPEN' FOR AN EXISTING INDEXED FILE. 'OPEN'
        * WILL SET UFBGKSIZE TO ZERO. THE USER'S PROGRAM MAY SET IT NON-ZERO
        * BEFORE A 'START' FUNCTION. 'START' WILL ZERO IT AGAIN. THE
        * USER'S PROGRAM MUST NOT MODIFY ANY OTHER FIELDS THAN
        * UFBGKSIZE IN THIS SECTION WHILE THE FILE IS OPEN.
        * **********************************************************************
```

```
00008C UFBKEYPOS                DS  H           KEY POSITION IN LOGICAL RECO
00008E UFBKEYSIZE               DS  HL1         KEY SIZE IN BYTES
00008F UFBGKSIZE                DS  HL1         GENERIC KEY LENGTH OVERRIDE
       *                                    MAY BE SET BEFORE 'START';
       *                                    USED ONLY BY 'START' FUNCTION;
       *                                    RESET TO BINARY 0 BY 'OPEN' AND
       *                                    EVERY 'START' FUNCTION
000090 UFBHXBLK                 DS  FL3         HIGHEST-LEVEL INDEX BLOCK
       *                                    ADDRESS FOR KEYED ACCESS
000093 UFBDABLK                 DS  FL3         FIRST DATA BLOCK ADDRESS
       *                                        (CURRENTLY ALWAYS 0)
000096 UFBPKI                   DS  H           INDEX ITEMS PER BLOCK
       *                                    FOR OUTPUT MODE
000098 UFBPTRD                  DS  FL4         FIRST BLOCK BEYOND
       *                                    PRIMARY EXTENT
       *                                    (INDEXED FILES)
00009C UFBPTRI                  DS  F           NEXT AVAILABLE INDEX
       *                                    BLOCK WITHIN PRIMARY EXTENT
       *                                    INDEX AREA
0000A0 UFBPTRN                  DS  F           NEXT AVAILABLE INDEX
       *                                    OR DATA BLOCK IN A SECONDARY
       *                                    EXTENT (INITIALLY ZERO)
0000A4 UFBBCBIOUT               DS  BL16         BCB FOR INDEX CREATION,
       *                                     OUTPUT MODE
0000B4 UFBPKD                   DS  H           RECORDS PER BLOCK FOR
       *                                     OUTPUT MODE
0000B6 UFBSPAREINX              DS  XL2          (RESERVED)
       UFBINXDISKEND            EQU *
       UFBINXDISKLGTH           EQU (UFBINXDISKEND-UFBBEGIN)
       * **********************************************************
       * MAGNETIC TAPE FILE EXTENSION SECTION:
       * FIELDS UFBTLABELS, UFBTDEN, UFBTSEQ AND UFBTFLAGS MAY BE SET
       * BEFORE 'OPEN' TO REQUEST OUTPUT LABELING OPTIONS, DENSITY
       * AND FILE POSITIONING.
       * ALL RELEVANT FIELDS AND FLAGS NOT SET BEFORE 'OPEN' ARE SUPPLIED
       * HERE BY 'OPEN' PROCESSING AND MAY BE EXAMINED BY THE USER'S
       * PROGRAM.
       * **********************************************************
0000B8                          ORG UFBDMSEND
00008C UFBTSPARE1               DS  BL4          (RESERVED)
000090 UFBTBCB                  DS  BL16         ADDITIONAL BUFFER CONTROL
       *                                        BLOCK FOR TAPE DOUBLE
       *                                        BUFFERING
0000A0 UFBTLABELS               DS  BL1          REQUESTED LABELING (OUTPUT)
       *                                        OR LABEL TYPE ON TAPE
       *                                        (INPUT)
       UFBTLABELSNL             EQU X'01'        UNLABELLED
       UFBTLABELSANY            EQU X'02'        ANY TYPE OF LABEL
       UFBTLABELSAL             EQU X'04'        ASCII LABELS
       UFBTLABELSIL             EQU X'08'        IBM LABELS
0000A1 UFBTDEN                  DS  BL1          TAPE DENSITY
       UFBTDEN800               EQU X'01'        800 BPI
       UFBTDEN1600              EQU X'02'        1600 BPI
```

```
        UFBTDEN556              EQU X'03'       556 BPI                 02↑
0000A2  UFBTSEQ                 DS  H           TAPE FILE SEQUENCE NUMBER
        *                                       (SET BEFORE OR DURING
        *                                       OPEN TO REQUEST POSITIONING
        *                                       AND AVAILABLE AFTER OPEN)
0000A4  UFBTFLG                 DS  BL1         TAPE-RELATED FLAGS
        UFBTFLGALLOWNL          EQU X'80'       *** OBSOLETE ***
        UFBTFLGSWITCH           EQU X'40'       TAPE VOLUME SWITCH REOPEN
        *                                       IN PROGRESS
        UFBTFLGEODEOV           EQU X'20'       TAKE EOV1 TRAILER LABEL AS
        *                                       EOF1 LABEL
        UFBTFLG7TRACK           EQU X'10'       USE 7 TRACK TAPE DRIVE FOR
        *                                       THIS FILE
        UFBTFLGNOHDR2           EQU X'08'       NO HDR2 FILE LABEL      01↑
0000A5  UFBTVOLSEQ              DS  BL1         TAPE VOLUME SEQUENCE NUMBER
        *                                       (ORDER OF VOLUME IN A
        *                                       MULTIPLE VOLUME FILE)
0000A6  UFBTSAVEVOL             DS  CL6         VOLUME NAME OF FIRST
        *                                       VOLUME OF A MULTI-VOLUME
        *                                       FILE SAVED HERE
0000AC  UFBTPARITY              DS  BL1         TAPE PARITY (7 TRACK TAPE
        *                                       ONLY)
        UFBTPARITYODD           EQU X'01'       ODD PARITY
        UFBTPARITYEVEN          EQU X'02'       EVEN PARITY
0000AD  UFBTSPARE2              DS  BL11        (RESERVED - MUST BE 0)
        UFBTAPEEND              EQU *
        UFBTAPELGTH             EQU (UFBTAPEEND-UFBBEGIN)
        * **********************************************************************
        * ADMS DISK FILE EXTENSION SECTION:
        * **********************************************************************
0000B8  UFBADMSFLG              DS  OBL1        ADMS FLAG
        UFBADMSFLGRS            EQU X'80'       RESTART FILE
        UFBADMSFLGBI            EQU X'40'       KEEP BEFORE IMAGE
        UFBADMSFLGAI            EQU X'20'       KEEP AFTER  IMAGE
        UFBADMSFLGCRASH         EQU X'10'       SOFT OR HARD CRASH OCCURED
        *                                       ON THIS FILE
        UFBADMSFLGCUPD          EQU X'08'       INDICATE CONSECUTIVE FILE
        *                                       SIZE CHANGED
        UFBADMSFLGEXTCK         EQU X'04'       INDICATES FIRST WRITE   03↑
        *                                       TO EXTENDED FILE        03↑
0000B8  UFBMCCPTR               DS  A           POINTER TO MCC CONTROL
        *                                       BLOCK (MAP-CONVERSION-CHECK)
0000BC  UFBMCC#                 DS  XL2         # OF MCC ENTRIES
0000BE  UFBSCHEMAID             DS  XL2         SCHEMA ID
0000C0  UFBVIEWID               DS  XL2         VIEW ID
0000C2  UFBRECORDID             DS  XL2         RECORD TYPE ID
0000C4  UFBADMSLOG              DS  0XL1        RECORD IMAGE LOG TYPE ON
        *                                       THE AUDIT TRAIL FILE
0000C4  UFBADMSSPB              DS  A           RESERVED USE BY SHARING
        *                                       TASK ONLY.  ADDRESS OF THE
        *                                       SHARED FILE POSITION BLOCK
        *                                       (SPB) OF THIS USER/FILE
```

```
0000C8 UFBADMSREC              DS  A            ADDRESS OF THE DEFAULT
       *                                        RECORD FOR THIS SCHEMA
       UFBADMSEND              EQU *
       UFBADMSLGTH             EQU (*-UFBBEGIN)
       * ************************************************************************
       * RESTART DISK FILE EXTENSION SECTION:
       * ************************************************************************
0000CC                         ORG UFBADMSFLG
0000B8                         DS   XL1
       UFBADMSFLGND            EQU X'40'        RESTART FILE EMPTY
       UFBADMSFLGNEXT          EQU X'20'        RESTART FILE READ NEXT
       UFBADMSFLGPRIOT         EQU X'10'        RESTART FILE READ PRIOR
0000B9 UFBRSUSCNT              DS  XL1          COUNT OF RESTART USER
0000BA UFBRSCURR               DS  XL2         RESTART CURRENCY POINTER 05↑
0000BC UFBRSFILENAME           DS  CL8          PSEUDO-RESTART FILE NAME
0000C4 UFBRSSPAR2              DS  BL8          (RESERVED)
       UFBRSEND                EQU *
       UFBRSLGTH               EQU (*-UFBBEGIN)
       UFBEND                  EQU *
```

```
                    VOL1
000000 VOL1              DSECT
       *             THE VOL1 RECORD IS THE STANDARD VOLUME LABEL FOR DISK OR
       *             MAGNETIC TAPE. ALL FIELDS ARE IN ASCII CHARACTERS.EXCEPT THE
       *             FDIR EXTENTS AND CREATION DATE. THIS RECORD ON DISK IS AT
       *             ADDRESS F'1', FOLLOWING THE IPL TEXT RECORD.
       *
       *             DATE  11-12-74
       *             VERSION 1.01
       *
       VOL1BEGIN                  EQU *
000000 VOL1ID                     DS  CL4'VOL1'    CHARACTERS 'VOL1'
000004 VOL1SER                    DS  CL6          VOLUME SERIAL NUMBER
00000A VOL1ACCESS                 DS  C' '         FILE PROTECTION CLASS
       *                                           OR BLANK
00000B VOL1RESRV1                 DS  BL26         RESERVED - ASCII BLANKS
000025 VOL1CREATOR                DS  CL3          FILE CREATOR ID OR BLANKS
       *                                           FOR MAGNETIC TAPE ONLY
000028                            ORG VOL1CREATOR
000025 VOL1OWNER                  DS  CL14         OWNER ID (OPTIONAL)
       *                                           FOR DISK AND TAPE VOLUMES
000033 VOL1RESRV2                 DS  BL28         RESERVED - ASCII BLANKS
00004F VOL1LEVEL                  DS  CL1'1'       MUST BE AN ASCII '1' FOR TAP
       VOL1TAPEEND                EQU *
       VOL1TAPELENGTH             EQU VOL1TAPEEND-VOL1BEGIN
000050 VOL1SYSTEM                 DS  CL8          SYSTEM IDENTIFICATION
000058 VOL1CREDATE                DS  PL3          VOLUME INITIALIZATION DATE
       *                                (PACKED YYDDD+)
00005B VOL1X1STRT                 DS  FL3          VOLUME TABLE OF CONTENTS 1ST
       *                                EXTENT STARTING BLOCK ON
       *                                VOLUME FROM 0
00005E VOL1X1END                  DS  FL3          FDIR 1ST EXTENT ENDING BLOCK
       *                                PLUS 1
000061 VOL1X2STRT                 DS  FL3          VOLUME TABLE OF CONTENTS 2ND
       *                                EXTENT STARTING BLOCK ON
       *                                VOLUME FROM 0
000064 VOL1X2END                  DS  FL3          FDIR 2ND EXTENT ENDING BLOCK
       *                                PLUS 1
000067 VOL1X3STRT                 DS  FL3          VOLUME TABLE OF CONTENTS 3RD
       *                                EXTENT STARTING BLOCK ON
       *                                VOLUME FROM 0
00006A VOL1X3END                  DS  FL3          FDIR 3RD EXTENT ENDING BLOCK
       *                                PLUS 1
       * (EXTENTS 2 AND 3 RESERVED. X2STRT THOUGH X3END MUST CONTAIN
       * BINARY ZEROES.)
00006D VOL1RESRV3                 DS  BL2          RESV'D IN DISK VOL1 LABEL 0↑
00006F VOL1UCBTYPE                DS  AL1          UCB TYPE                  0↑
000070 VOL1VCBBC                  DS  AL2          BLOCKS PER CYLINDER       0↑
000072 VOL1VCBMAXTFR              DS  AL2          MAX TRANSFER (BYTES)      0↑
000074 VOL1VCBCV                  DS  AL2          CYLINDERS PER VOLUME      0↑
000076 VOL1VCBCVP                 DS  AL2          CYLINDERS PER PHYS VOLUME 0↑
000078 VOL1RESRV4                 DS  BL136        RESV'D IN DISK VOL1 LABEL 0↑
       *                                ASCII BLANKS
       VOL1DISKEND                EQU *
       VOL1LENGTH                 EQU 256
```

CHAPTER 6: SUPERVISOR CALLS

6.1 INTRODUCTION

This chapter describes the input parameters, outputs, and function of each supervisor call (SVC) which is directly accessible to normal user programs. SVC descriptions are ordered by SVC number, beginning with SVC 0. The following list is a summary of all available SVC's. Those with names in parentheses are normally not accessible to user programs and so are not described in this manual.

Name	Number	Residency	Inner SVC Only
OPEN	SVC 0	(NR)	
CLOSE	SVC 1	(NR)	
TIME	SVC 2	(NR)	
XIO	SVC 3	(R)	
LINK	SVC 4	(NR)	
GETBUF	SVC 5	(NR)	
FREEBUF	SVC 6	(NR)	
(WAIT)	SVC 7	(R)	I
(SEND)	SVC 8	(R)	I
(FIX/UNFIX)	SVC 9	(NR)	I
(GETMEM)	SVC 10	(R)	I
(FREEMEM)	SVC 11	(R)	I
HALTIO	SVC 12	(NR)	
(DTI)	SVC 13	(R)	
ALEX	SVC 14	(NR)	
UNLINK	SVC 15	(NR)	
CANCEL	SVC 16	(NR)	
CHECK	SVC 17	(R/NR)*	
(SEIZE/RELEASE)	SVC 18	(R)	I

* 'Intervention required' handling and I/O error logging facilities of CHECK may be non-resident.

READVTOC	SVC 19	(NR)	
GETPARM	SVC 20	(NR)	
(GETDISK)	SVC 21	(NR)	I
(FREEDISK)	SVC 22	(NR)	I
(CREATFDR)	SVC 23	(NR)	I
READFDR	SVC 24	(NR)	
(UPDATFDR)	SVC 25	(NR)	I
RENAME	SVC 26	(NR)	
SCRATCH	SVC 27	(NR)	
EXTRACT	SVC 28	(NR)	
(PLEASE)	SVC 29	(NR)	
MOUNT	SVC 30	(NR)	
PCEXIT	SVC 31	(NR)	
SETIME/RESETIME	SVC 32	(NR)	
PUTPARM	SVC 33	(NR)	
(MWAIT)	SVC 34	(R)	I
SET	SVC 35	(NR)	
XMIT	SVC 36	(R)	
CREATE	SVC 37	(NR)	
DESTROY	SVC 38	(NR)	
CEXIT	SVC 39	(NR)	
GETDIAG/FREEDIAG	SVC 40	(NR)	
DISMOUNT	SVC 41	(NR)	
PROTECT	SVC 42	(NR)	
LOGOFF	SVC 43	(NR)	
reserved	SVC 44		
(LOADCODE)	SVC 45	(NR)	

Where input parameters to an SVC routine are supplied on the stack, they are removed before returning to the issuer, leaving only the specified output values.

Open File

OPEN (SVC 0) NON-RESIDENT

Inputs: The top word of the stack addresses a User File Block (UFB) containing:

- Reference name (UFBPRNAME)
- Optional parameter indicators (UFBF1)
- Mode flags (UFBF2)
- Device class (UFBDEVCLASS)
- Device dependent flags (UFBF4)

A new workstation file or printer file must contain:

- File organization (UFBFORG)
- Logical record size (UFBRECSIZE)

A new indexed disk file must contain:

- Key position (UFBKEYPOS)
- Key size (UFBKEYSIZE)

A new magnetic tape file must contain:

- Physical block size (UFBBLKSIZE)
- Tape sequence number (UTBTSEQ)

And a new disk file must contain:

- Disk space requirement UFBNRECS

A new or existing file may optionally contain:

- Complete actual file name and volume (UFBVOLSER,UFBDIRNAME,UFBFILENAME), or device number, or unqualified name of member within library or volume (UFBFILENAME). For an existing tape or disk file, if UFBFORG or UFBRECSIZE are supplied, they must agree with the values in the file label. Also, for tape files, either filename must be supplied or UFBTSEQ must be non-zero. For disk files, UFBBUFSIZE may be supplied as the maximum buffer size required.

Fields which may be preset, but are not used during 'OPEN' (except for TC files):

- UFBERRAD, UFBEODAD, UFBRECAREA, UFBKEYAREA.

For TC files, UTBRECAREA contains the address of the TC connection parameters.

The high-order byte of the top word of the stack may be set for the OPEN exit option as described below.

Outputs: Open File Block (OFB) created in protected memory. Device indication placed in OFB (OFBUCB). Mode flags copied to OFB from UFB (UFBF2). File Location and Use Block (FLUB) created for disk file if not already present (for another task). Extent information placed in FLUB for disk files. Length of extent list placed in FLUB for disk files. File Descriptor Record (FDR1) block and record numbers placed in FLUB for disk files. Sharing mode flags set in FLUB.

Addresses of I/O function processing routines placed in UFBVREAD through UFBVSTART. Number of records in file placed in UFBNRECS for disk files or for magnetic tape files opened in EXTEND mode. Sequential record pointer initialized to first record of file (last record of file plus one record if EXTEND mode). Buffer addresses and lengths placed in buffer control fields of UFB (UFBBUFADR, UFBBUFSIZE), and these buffers marked 'contents not valid' (in UCBBCBFLAGS) and 'buffer in protected memory' when required. The file status bytes (UFBFS1, UFBFS2) are set = '00'; UFBLF is set = Open.

For existing files only:
- File organization indicated (UFBFORG)
- Logical record size supplied (UFBRECSIZE)
- Block size supplied (UFBBLKSIZE)

For disk files only:
- File attribute flags indicated (UFBFLAGS)
- Record size from label supplied (UFBLRECSAVE)
- Last block number in file supplied (UFBEBLK)
- Last record number in this block supplied (UFBEREC)
- Number of 2K blocks within file extents supplied (UFBNBCKS)

For indexed disk files only:
- First data block number (relative - within file) supplied (UFBDABLK)
- Highest-level index block number (relative - within file) supplied (UFBHXBLK)

For existing indexed disk files only:
- Key position supplied (UFBKEYPOS)
- Key size supplied (UFBKEYSIZE)

The Volume Control Block (VCB) for a volume containing a disk file is updated to indicate that an additional file is open on this volume (VCBOCNT).

For disk or tape files, buffers are allocated from the user-modifiable segment and their allocation recorded in the task's extended Task Control Block. For disk or tape files, no buffers are allocated when physical access method functions are used (UFBF1PAM set).

Program-supplied file, volume, and device information (as supplied through the 'Open Parameters' sections of the UFB) remains in the UFB, or is replaced with information acquired by OPEN through GETPARM.

For device = dummy, an OFB is set up to indicate that a null file has been opened. There is no IORE, and OFBIOREPTR and OFBUCB are zero. A null file is valid in INPUT mode only and all DMS read requests to the file return with file-status='10' (end-of-file). UFBRECSIZE and UFBFORG are not set by OPEN for a null file.

For device = disk and UFBF4NOVTOC set, a disk volume without a Volume Table of Contents (VTOC) is indicated. In this case, the unique filename in the FLUB is created by setting FLUBFILENAM1 and FLUBFILENAM2 = blanks; FLUBFDRR and FLUBFDRB are zero; and the volume (through FLUBVCB) will have VCBFLAGSNOVTOC set. The FLUB contains one extent which allows access to all blocks on the volume. For NOVTOC, the file organization is consecutive and the record size is 2K unless the user program supplies a record-size. Extend and Shared Modes are not supported for NOVTOC volumes.

For output mode, UFBF4RLSE is set. This setting may be overridden by the user when the OPEN-GETPARM is issued or the bit may be cleared by the program after OPEN. The bit causes unused space in the file to be released by CLOSE.

The OPEN exit option is available by setting the high-order byte of the top word of the stack. This byte can indicate the conditions listed below. If a condition arises for which the corresponding bit is set, then control returns to the program at the next instruction in sequence, rather than SVC OPEN issuing a GETPARM (RESPECIFY). If the exit is taken, UFBFS1 = X'39', UFBFS2 = a mask for the appropriate condition; and UFBPREVO is set. The SVC OPEN may be reissued after an OPEN exit has been taken.

OPEN exit conditions:

UFBFS2XFILE (X'80') - Return if the file is not found (non-OUTPUT) or if a duplicate file name is found (OUTPUT mode).

UFBFS2XLIB (X'40') - Return if the library is not found (non-OUTPUT mode).

UFBFS2XVOL (X'20') - Return if the volume is not mounted.

UFBFS2XSPACE (X'10') - Return if there is insufficient space on the volume for a new file (OUTPUT mode).

UFBFS2XVTOC (X'08') - Return if there is no VTOC space on the volume (OUTPUT mode).

UFBS2XTAPELD (X'08') - Return tape label type or tape density is not acceptable to the program.

UFBFS2XPOS (X'04') - Return for possession conflict, which includes file already open by current program, file opened by other program and open modes conflict and volume possession is exclusive for another user.

UFBFS2XPROT (X'02') - Return if the user does not have access rights required to open the file.

UFBFS2XFORMAT (X'01') - Return if there is an error in specification of file format. Error class is described in UFBXCODE.

A NO-MESSAGE option is also available such that control will return to the program (at the next instruction in sequence) whenever any condition arises for which a RESPECIFY (or CANCEL) message would normally be returned. If UFBF4NOMSG is set, then the MSG-ID (4 bytes) of any RESPECIFY or CANCEL message is stored in the first 4 bytes of the UFB and UFBFS1=X'36' and UFBFS2=X'0'. The NO-MESSAGE option is checked after any open-exit checking is performed. (Open-exits provide a more exact indication of the problem; the NO-MESSAGE option is useful for tasks with special processing requirements.)

Note 1: This list of indirect outputs of 'OPEN', although extensive, is not intended to be completely exhaustive.

Note 2: When creating an indexed file in OUTPUT mode, the 'packing density' for both index blocks and data blocks may be set by the user. For example, if 20 records at 100 bytes each would normally fit into a data block, then at 80 percent packing, each data block would be loaded with only 16 records. UFBPKD (data) and UFBPKI (index) are settable by the user before Open; OPEN will use the binary value as a percentage. A value of 50-100 will be accepted as the percentage value for packing. Any value outside this range will be ignored; the default used will produce full packing. UFBPKD and UFBPKI should not be modified after OPEN.

Note 3: For OUTPUT MODE DISK FILES, UFBOUTRECS contains the record count used when allocating file space. This value is available immediately after SVC OPEN. UFBNRECS=0 after OPEN.

Note 4: A complete description of Work files and Print files (including SVC OPEN processing) is included in Chapter 2 of this document.

Function: Prepares a file for processing by Data Management System functions. Information from the file label is brought into memory. Devices and volumes are allocated and reserved as required. Control blocks are set up. Buffer space is allocated. The OPEN function may interface with a conversational user's workstation to request information which has not been supplied.

Close File

CLOSE (SVC 1) NON-RESIDENT

Inputs: The top word of the stack addresses (open) User File Block (UFB). The high-order byte of this word may have the following flag bits set:

Bit 0 = 1 Force end-of-volume REEL.
Bit 1 = 1 No rewind (magnetic tape).
Bit 2 = 1 Rewind and unload (magnetic tape).

The other bits of this byte must be zeroed.

Outputs: Open File Block (OFB) in protected area is deallocated, and unchained from the task's TCB (OFBTASKCHN) and the FLUB for the file (OFBFILECHN). File Location and Use Block (FLUB) in protected area is deallocated if the file is no longer open to any task.

Volume Control Block (VCB) for volume containing disk file is updated to indicate that the file has been closed (VCBOCNT). Unit Control Block (UCB) for nonshared devices marked 'deallocated'.

Any records which remain to be written from buffers are written (as specified by UFB buffer control fields), and then these buffers are deallocated and marked as such in the Task Control Block (TCBAVBUF).

For a TC file, a DISCONNECT operation is performed for the associated line if no other files remain open which are associated with that line.

For disk and tape files, the file label is rewritten if the file was being processed in IO, SHARED, OUTPUT or EXTEND modes (FDR1 for disk, trailer labels for tape). The updated number of records in the file is placed in this label (from UFBNRECS or as calculated from UFBEBLK and UFBBUFOFFSET).

For OUTPUT mode (consecutive files only), unused blocks in the file are released when UFBF4RLSE is set.

For disk files, these additional label fields are updated when required:

Date of last modification (FDR1MODDATE)
Last record's block within file (FDR1EBLK)
Last record's number in block (FDR1EREC)
Highest-level index block number (FDR1HXBLK)
First data block number (FDR1DABLK)

Function: Removes a file from processable status. Information remaining in memory is written to the secondary storage device containing the file or discarded. Memory areas allocated for processing this file are released and may be used for other purposes. The User File Block (UFB) is returned to a state in which another OPEN can be issued on the file. The file status reflects the last operation performed on the file by Close (if any) or by DMS.

Get Date and Time

TIME (SVC 2) NON-RESIDENT

Inputs: The top word of the stack contains a binary 0 if the time is to be returned in hours, minutes, seconds, and hundredths of seconds (packed decimal), or a binary 1 if the time is to be returned in hundredths of seconds (from midnight) binary.

The next word contains a binary 0 if the Julian day is to be returned, or a binary 1 if the date as year-month-day is to be returned.

Outputs: Time returned in the top word of the stack as either:

(a) Packed decimal 'HHMMSSth' where HH is hour (on 24-hour clock), MM is minute-in-hour, SS is second-in-minute, t is tenth of second and h is hundredth of second, or

(b) Binary hundredths of second from preceding midnight.

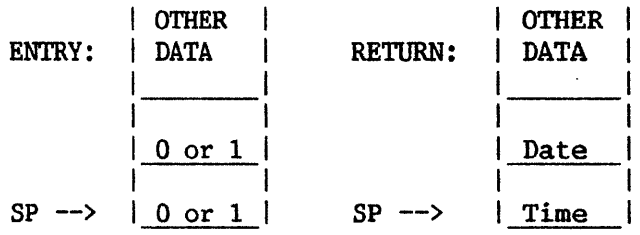
Date returned in the next word as packed digits in one of these formats:

(a) '00 YY DD DF', where YY is year-in-century, DDD is day-in-year, F is hexadecimal 'F' for unpacking.

(b) '0Y YM MD DF', where YY is year-in-century, MM is month-in-year, DD is day-in-month, F is hexadecimal 'F' for unpacking.

Function: Supplies the current date and time to users' programs or system routines in the forms described above.

Stack on entry and return:



Execute Physical I/O

XIO (SVC 3) RESIDENT

Inputs: The top 16 bytes of the stack are a parameter list containing:

Byte 0: Flags

- Bit 1 = 1 - Special block-on-volume-oriented disk I/O request - "VOLIO" - valid only when requested by system routines in System Mutual Exclusion state or when the accessed volume is mounted for initialization by the issuing task.
- Bit 2 = 1 - Block print operation. Data must be 2K-aligned and is not moved to the device's resident print buffer.
- Bit 3 = 1 - 'Halt I/O Queue' option (for Disk Mount operation).
- Bit 4 = 1 - Reset UCBSTATNOTOP and UCBSTATNOCODE to allow I/O device being simulated to malfunction.
- Bit 5 = 1 - Issue Return Code = 32 if I/O issued to inoperative workstation.
- Bit 6 = 1 - Force uppercase printing.
- Bit 7 = 1 - Telecommunications Option--TRANSMIT or RECEIVE.

Bytes 1-3 - Address of Open File Block (OFB) for file with the following exceptions: When flag bit 7 = 1 (TC option), the OFB address is for the VS-DLP communication path; when flag bit 1 = 1 (VOLIO option), the address of the Volume Control Block (VCB) for the disk volume.

Byte 4 - Command byte for I/O Command Word (IOCW).

Bytes 5-7 - Memory address (virtual) for IOCW if read or write command in byte 4.

Bytes 8-9 - Length in bytes for operation (read or write command, all devices).

Bytes 10-11 - Unused.

Bytes 12-14 - For disk files only. Block number within file (in binary) of the first block to be read or written (where the first block of a file is block 0). With the following exceptions: If VOLIO option is selected, the block number within the volume (in binary) of the first block to be read or written (where first volume block is block 0); if TC option is selected, bytes 12-14 contain bytes 6-8 of the IOCW.

Byte 15 - Unused

Outputs: Return codes in the top word of the stack (replacing the input parameters).

Low-order halfword of return code field - binary return codes:

0 - Success
4 - Truncation at end-of-extent (non-VOLIO disk only)
8 - Truncation at end-of-cylinder or end-of-track (disk only)
12 - Starting block number beyond end-of-file (non-VOLIO disk) or beyond end-of-volume (VOLIO disk)
16 - Invalid data address or data length. (Data address for disk must be page-aligned; for other devices word-aligned. Virtual memory area encompassed by the area from data address through data-address plus block-size-minus-one must either be in the segment 2 I/O buffer area or entirely above the XIO parameter list on the stack if the XIO is issued from unprivileged state. The specified length must not imply spanning of more pages than there are Indirect Address List entries for the device.)
20 - Second XIO on file without intervening CHECK
24 - TC XIO attempted on an OFB that was not created as the result of an 'IPOPEN' on an IPCB.
28 - TC XIO attempted on a device reserved exclusively by another task.
32 - XIO has been issued to inoperative workstation and I/O has not been issued (bit 5 of option flag must be set for issuance of this return code).
36 - TC XIO attempted on a peripheral processor (DLP) reserved exclusively by another task.

High-order halfword of return code field - residual block counts:

Return codes 0, 12, 16, 20 - Always zero.
Return codes 4, 8 - Specified block size minus number of bytes actually read or written.

Note: If return codes 0, 4, or 8 are set, the I/O operation is queued for initiation and a CHECK must be issued to test for completion. If return codes 12, 16, or 20 are set, the operation has been suppressed. XIO never waits for I/O completion.

Function: Initiates I/O operations for the Data Management System, as requested by its parameter list.

The following restrictions on general I/O capability are enforced by the XIO routine:

All - (1) All memory addresses for a read or write operation must be valid (present in main memory or page faulted) and, unless the requesting routine is privileged, must be in the user-modifiable segment and either:

(a) in the I/O buffer area, or
(b) entirely above the XIO parameter list on the stack.

Disk - (1) A block to be read or written must fall within the current extent limits of the specified file (except for "VOLIO" disk requests).

(2) The specified memory address must be on a page boundary.

(3) The "VOLIO" option (flag bit 1 = 1) is allowed only when requested from within System Mutual Exclusion state.

(4) The length specified for a READ or WRITE operation must be a multiple of the page size.

(5) Indirect data addressing is always used for disk I/O.

Diskette - Library-Structured

(1) All restrictions as for other disk.

Diskette - Unstructured

(1) A block to be read or written must fall within the bounds of the diskette platter (blocks 0 through 153); otherwise, return code 16 is set.

(2) The "VOLIO" option is ignored.

NOTE: A non-standard addressing option is now supported that allows the user to format a soft-sectored diskette in any combination of sector size and density. The use of this option is intended to be limited to specialized utilities. User programs which employ this option are responsible for performing direct and sequential I/O on a physical-sector basis. The user program must calculate the sector size and addresses, set mode, and set density. When non-standard addressing is specified,

the XIO SVC will not perform extent validation or address translation, but simply passes the address to the firmware via the I/O Control Word (IOCW).

Tape - (1) Maximum size permitted for tape records is 32K bytes.

Printer-

(1) Bit 2 of the first byte of the XIO parameter list distinguishes between print operations through a resident print buffer and multiple-line ('block') print operations. The data length for single-line print operations must be not less than 2 or more than 134. The data for a block print operation must be on a single page.

(2) The data for a block print operation must include record length bytes. The data for single-line print operations through a resident buffer should include only the printer control characters and the characters to be printed.

Workstation

(1) An Attention Identification (AID) character is stored in the current status portion of the device Unit Control Block (UCB) on successful completion of each I/O operation. (Refer to VS Principles of Operation Manual or VS Operating System Services Pocket Guide for a listing of these characters.) The AID character also serves to indicate whether the workstation keyboard is in locked or unlocked state after the operation.

(2) If the device's UCB indicates 'Keyboard unlocked' when a READ operation is requested, the XIO routine will wait for an attention interruption from this device. When such an interruption is received, the interrupt service routine will mark the UCB 'Keyboard locked' and then will allow XIO to initiate the read operation.

(3) Indirect data addressing is always used by XIO for workstation I/O.

Link to Another Program

LINK (SVC 4) NON-RESIDENT

Inputs: The top 16 or 32 bytes of the stack contain a parameter list as follows:

Byte 1 - Flags, as follows:

- Bit 0 = 1 Search only system library for member (see below).
- Bit 1 = 1 Report failure to find the specified member, member not executable, member already opened (other than Shared Read-only), or password for member or directory not provided, or other error by returning to location of SVC instruction plus 6 (next instruction plus 4), rather than by entering the Help Processor. See outputs below for return codes in these cases.
- Bit 2 = 1 Branch to location of LINK SVC plus 10 (next instruction plus 8) after setting up to enter new program or subprogram, with entrypoint address of new program in register 0. (See LOADONLY operand description of LINK macro in Chapter 4.)
- Bit 3 = 0 Must be zero.
- Bit 4 = 1 Overriding user program library and volume specified in parameter list bytes 17-30.
- Bits 5-7 Must be zero.

Bytes 2-9 - Character string (ASCII) containing name of program to be LINKed to. This name is used with the current program library specification as a member name within a library or volume, and the resulting complete filename is sought. If such a file is not found, the member name is used with the system library specification, and the resulting complete filename is sought. If flag bit 0 above is set, only the system library specification is used. If flag bit 1 is not set, failure to find the program in either library causes abnormal termination of the issuing task.

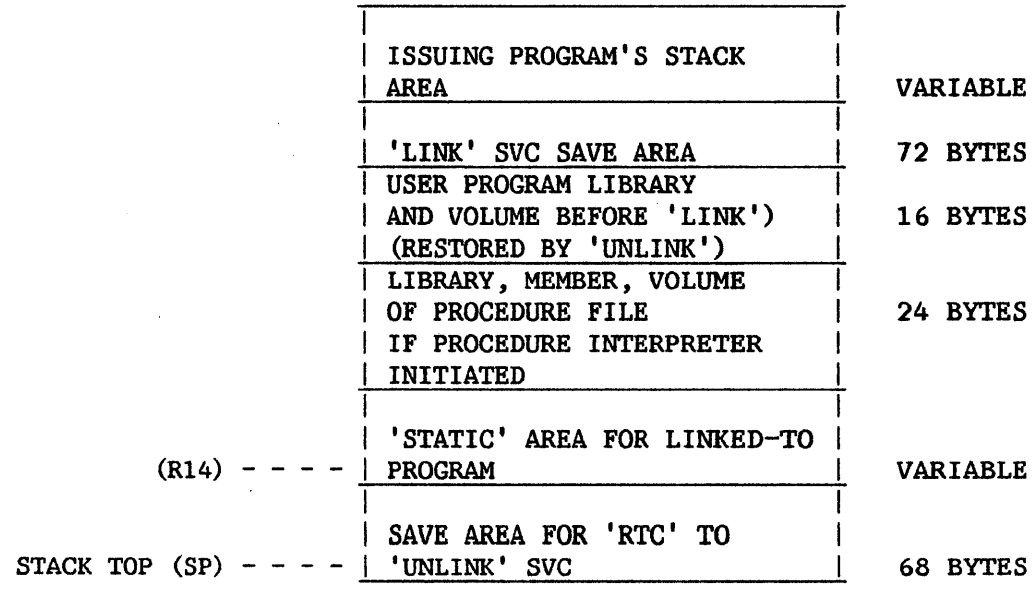
Bytes 10-16 - Not examined.

Bytes 17-22 - Character string containing overriding user program library's volume serial. Totally absent unless flag bit 4=1.

Bytes 23-30 - Character string containing overriding user program library. Totally absent unless flag bit 4=1.
Bytes 31-32 - Not examined. Absent unless flag bit 4=1.

Register R1 by convention addresses a standard argument list to be passed to the program invoked as a result of the LINK.

Outputs: A new Program File Block is built and chained to the active TCB, indicating that a program has been entered. A new Program Exception Element is built and enstacked from TCBPXE, indicating that no program exception exits are set. Register R14 addresses the first (doubleword-aligned) address of the newly created group of 'static' areas (see below). Register R1 addresses the argument list passed by the issuer (if provided). Other register contents (except register 15) are unchanged. Static storage areas for the LINKed-to program are pushed onto the stack. A single word of control information (address of UNLINK' routine to be executed on RETURN) follows. This is followed by a save area chain word and register save area suitable for use with the RETURN macroinstruction (RTC instruction). Control register 1 contains the address of this save area (as does general register 15) on completion of the LINK. The stack on entry to the LINKed-to program is as follows:



If no linkage can be made to the specified member, the above does not occur; but if parameter byte 1, bit 1 was set, control passes to the location of the LINK SVC instruction plus 6 with binary return codes in the top

word of the stack (replacing the input parameters) as follows:

Return code = 0 - Not a program file, and the procedure interpreter cannot be invoked.
Return code = 4 - Volume not mounted
Return code = 8 - Volume in exclusive use by another user
Return code = 12 - All buffers in use when one was required by LINK
Return code = 16 - Directory not found
Return code = 18 - Access to program's file-protection class denied
Return code = 20 - File not found
Return code = 24 - Unused
Return code = 32 - FDX1 and FDX2 conflict detected by READFDR
Return code = 36 - FDX2 and FDR conflict detected by READFDR
Return code = 40 - Invalid parameter passed to READFDR (including NL volume type)
Return code = 44 - I/O error on VTOC
Return code = 48 - Unable to read FDR2 record (additional extent specifications)
Return code = 52 - Invalid program file; unable to complete LINK
Return code = 56 - File open other than shared read-only

Function: To pass control to a user-level program or subprogram not linkage-edited with the issuing program. Also used by the command processor to initiate execution of a requested program, or of the system's procedure interpreter.

User parameters to be passed to the invoked program must not be in the reentrant program area (segment 1).

'Static areas', as defined, are static only within a group of program modules linkage-edited together, and will be removed (popped) from the stack by execution of the 'UNLINK' supervisor call routine.

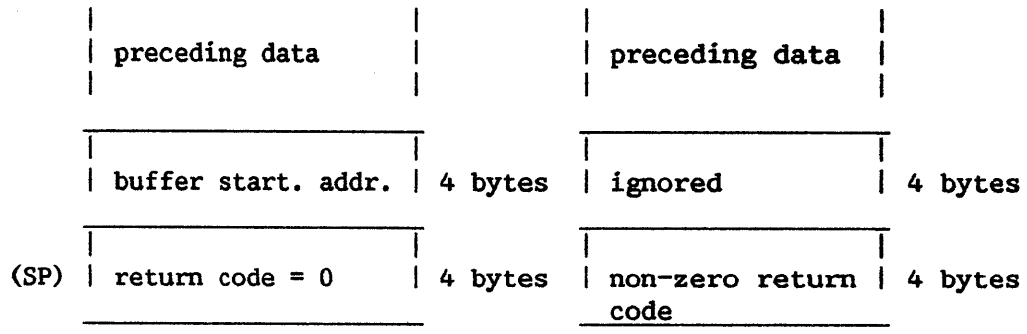
Get Buffer Space

GETBUF (SVC 5) NON-RESIDENT

Inputs: A 2-word parameter list on top of the stack. The lower-addressed word contains the buffer length requested, which must be a multiple of the page size (2048). The second word of the parameter list is ignored on entry.

Outputs: A zero return code and the starting address of the allocated buffer OR a nonzero return code placed on top of the stack:

Return code = 0 - A buffer area is allocated (Figure 1).
Return code = 4 - All buffers in use (Figure 2).
Return code = 8 - Requested length is not a multiple of 2K (Figure 2).



(Figure 1)

(Figure 2)

Function: To allocate a buffer area as requested. Buffer space is taken from the low address end of segment 2. Value in control register 2 (the stack limit) may be modified.

Free Buffer Space

FREEBUF (SVC 6) NON-RESIDENT

Inputs: Address of the buffer to be returned in the top word of the stack. Length of the buffer area to be returned in the next word above. This length must be a multiple of the page size (2048).

NOTE:
In the case of calls by the GETHEAP/FREEHEAP SVC, the high-order byte of the word specifying the length of the buffer area is set to X'04'. In the case of such calls, GETHEAP/FREEHEAP places the address of the corresponding subpool block (SPB) in the third word from the top of the system stack.

Outputs: A return code in the top word of the stack replacing inputs:

Return code = 0 - Buffer deallocated.
Return code = 4 - Invalid buffer address.
Return code = 8 - Invalid buffer length.

Function: To de-allocate a buffer area allocated by GETBUF (SVC 5). Value in control register 2 (the stack limit) may be modified.

Halt I/O Operation

HALTIO (SVC 12) NON-RESIDENT

(1) PRINTER OPTION (retained but made obsolete by "General Option", below):

Inputs: word on top of stack
byte 0: X'00'
bytes 1-2: ignored
byte 3: device number of a printer

Outputs: HIO instruction issued to the specified device, if a valid printer which is in use by the issuing task. If an "I/O processor busy" response is received, the corresponding Unit Control Block (UCB) is marked, and the HIO instruction reissued when an "I/O processor now ready" interrupt is received. The issuing task must still wait for any outstanding I/O to complete (see SVC CHECK). The condition code from the HIO instruction is returned in the word on the stack top.

Function: To speed termination of multiline print operations which are to be aborted.

(2) GENERAL OPTION

Inputs: word on top of stack
byte 0: X'80'
bytes 1-3: OFB address, or VCB address + 1

Outputs: none (parameters removed from stack).

HALTIO should be issued only if an XIO has been issued, but the CHECK has not been done. CHECK should be issued after HALTIO (as in a normal wait-for-completion).

Allocate Additional Extent

ALEX (SVC 14) NON-RESIDENT (No longer available.)

Inputs: The top word of the stack addresses an Open File Block (OFB) for the file.

Outputs: An additional extent is allocated for the given file. Subsequently the entries containing extent information in FLUB and FDR records of the file are modified to record the allocation. A return code is placed on top of the stack replacing the inputs:

Return code = 0 -	Additional extent allocated
Return code = 4 -	Invalid OFB address; no allocation
Return code = 8 -	File in INPUT mode; no allocation
Return code = 12 -	Wrong device type; no allocation
Return code = 16 -	Extent limit exceeded; no allocation
Return code = 20 -	All buffers in use; no allocation
Return code = 24 -	Volume full; no allocation
Return code = 28 -	No space in VTOC for FDR2; no allocation
Return code = 32 -	Disk I/O error; VTOC unreliable

Function: To allocate an additional extent for a disk file. The extent size to be allocated is found in FDR1SECEXT of the FDR1. If an extent of the size specified in Field FLUBSECEXT of the corresponding FLUB is not available on the volume, the largest available extent is allocated.

Note: A file has to be opened for exclusive use before ALEX is issued.

Return From Program Entered By Link

UNLINK (SVC 15) NON-RESIDENT

Inputs: Field ETCBPFB of the task's Extended Task Control Block (ETCB) addressing save area and control information for 'UNLINK' use as pushed onto the stack by LINK (SVC 4).

Outputs: The program which issued the LINK (SVC 4) to enter the issuing program is restored to the reentrant program area (segment 1) of the task's virtual address space. The previously active program's PFB and all its PFEs are released. UNLINK closes shared files OPENed on the link level it is destroying, if the user fails to do so. UNLINK releases any devices reserved on the level it is destroying, if the calling program fails to do so. The save area and control information is removed (popped) from the stack. The LINKed-to program's 'static' area is popped from the stack. UNLINK attempts to close those files which were opened at the Link level being unlinked (UNLINK determines which files should be CLOSEd by comparing a byte in the OFB (OFBLINKLEV), which was set at OPEN time, with the current link level being UNLINKed (ETCBLINKLEV)). Control is then passed to the program which issued the LINK, at the instruction address folllowing the LINK SVC.

Function: To return from a LINKed-to program. This SVC is normally issued by code in the system area (segment 0) entered as a result of a RETURN sequence in the LINKed-to program. The 'UNLINK' SVC may, however, be issued directly by a program.

Cancel Program

CANCEL (SVC 16) NON-RESIDENT

Inputs: Word on top of stack containing address of message to be displayed.

Outputs: Help Processor entered with no return to issuing program. Program abnormally terminated when the user issues the 'CANCEL' command to the Help Processor.

Function: To terminate a program in the event of uncorrectable program failure, such as:

(a) exhaustion of a system resource,

(b) illegal or invalid parameters to an SVC routine or other system service program,

(c) a program-detected condition which cannot be satisfactorily resolved within the program.

A standard message is generated to notify the user of the error situation. The supplied message is also displayed. The user cannot immediately resume program execution by the 'CONTINUE' command. He may, however, examine the program by means of the Help Processor's debugging facilities, modify the current instruction address by means of the 'CHANGE' command, and then attempt to resume program execution. A program terminated by a CANCEL SVC issued by another SVC routine cannot be continued.

The message at the specified address is in the following format:

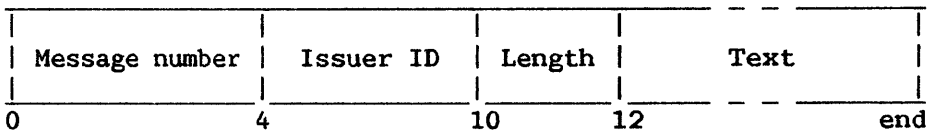
(1) Four-byte message number in ASCII characters. Always required.

(2) Six-byte issuer identification in ASCII characters.

(3) Two-byte message length in binary. This is the length of the text which follows as (4) here.

(4) Message text in ASCII characters. If the message is of more than one line, an end-of-line is indicated by an ASCII 'new line' character. No line may contain more than 79 characters, including the end-of-line indicator. The last (or only) line does not require an end-of-line character.

Message format:



Check For Event Occurrence

CHECK (SVC 17) PARTIALLY RESIDENT

Inputs: Eight-byte parameter items. For single-event CHECK, the single item is on top of the stack. For multiple-event CHECK, the top word of the stack contains the address of the list of items. The high-order byte of this word (bits 1-7) contains a count of the number of items, and bit 0 of this byte is set to 1; the next word on the stack is not examined in this multiple-event case. For multiple-event CHECK, if the option flag (byte 0) in the input parameter list for the event is set equal to X'FF', then the particular event is bypassed, i.e., no WAIT is done for the event.

The FORM=LIST option of the CHECK macro can be used to build a multiple-event CHECK list on the stack (see CHECK macro description for further details).

Each eight-byte item is as follows.

(1) Normal I/O check (OFB) item:

Byte 0	Zero.
Bytes 1-3	OFB address.
Bytes 4-7	Alternate return address to be used in case of I/O error, or zero. If the low-order bit of Byte 7 is on, then completion IOSW is returned in general registers 0 and 1.

(2) VOLIO I/O check (VCB) item:

Byte 0	Zero.
Bytes 1-3	VCB address plus 1.
Bytes 4-7	Alternate return address to be used in case of I/O error, or zero. If the low-order bit of Byte 7 is on, then completion IOSW is returned in general registers 0 and 1.

(3) Timer check item:

Byte 0	X'10'
Bytes 1-7	Ignored.

(4) Intertask message check item:

Byte 0	X'20'
Bytes 1-3	Address of an area in segment 2 in which to receive a message. The first two bytes of this area must contain its length in bytes (binary) including these bytes. This length must not be greater than 2016. The message (not including its length bytes) is

moved to the area following these bytes and truncated if too long for the specified area. The area's length bytes are adjusted to reflect the length of the message, including these bytes.

Bytes 4-7 The name (CL4) of one of this task's active message receipt ports, as established by CREATE.

(5) Workstation program function key check item:
Byte 0 X'40'
Bytes 1-3 Workstation device number
Bytes 4-7 Ignored

(6) Unsolicited interrupt:
Byte 0 X'08'
Bytes 1-3 Number of any device on line
Bytes 4-7 Address of 8-byte area to receive IOSW (must be in user-modifiable buffer area or in stack as validated by MCBRWTST)

(7) TC event:
Byte 0: X'01'
Bytes 1-3: The OFB address of the TC device on which XIO was issued
Bytes 4-7: The address of an eight-byte receiving area for the completion IOSW, or binary zeroes if the IOSW is not desired

Outputs: Single-Event CHECK)
Inputs popped from stack. For I/O completion CHECK, a workstation message is displayed if possible on 'intervention required' conditions. See the XMIT SVC description for the format of intertask messages after a 'message' CHECK.

Multiple-Event CHECK
One word of inputs popped from stack. Second word replaced by displacement within parameter item list of item corresponding to the event which has occurred. (Displacement is from 0, by 8.) Device 'intervention required' conditions must be handled by the CHECK issuer, who must re-issue a CHECK (single- or multiple-event) to wait for I/O completion.

Function:
(1,2) Waits for completion of an I/O operation initiated by means of the XIO supervisor call. In the event of a permanent error completion (IOSW bit EC set, bits NC or IRQ not set), returns to the alternate return

address. Otherwise returns to the next sequential instruction address. If 'intervention required' (IOSW bit IRQ) is indicated on completion, issues an appropriate workstation message if possible, expecting either an unsolicited interrupt from the device when it becomes ready, or a response when the condition has been corrected; may reissue the message if the condition has not been corrected; then waits for another completion indication. Writes an I/O error logging record in the event of a device malfunction or main memory parity error. The Volume Control Block address option is available only to routines in System Mutual Exclusion (SME) state, or which have exclusive use 'for initialization' of the disk volume to which the VCB refers. (The VCB option is intended for use in conjunction with the VOLIO option of the XIO supervisor call.)

(3) Waits for expiration of a timing interval. The issuing program is cancelled if no interval is set.

(4) Waits for receipt of a message directed to the specified port name, which must have been established by the issuing task by means of the CREATE SVC.

(5) Waits for a program function key to be struck at the specified workstation. The issuing program is cancelled if an un-CHECKed I/O operation (XIO) has been issued to the specified workstation, or if that workstation is not reserved for use by this task.

(6) Waits for an unsolicited interrupt from a workstation or printer. For a workstation, this CHECK option waits for a program function key whether the keyboard is locked or not. The issuing task will be cancelled if the device is not reserved for use by the issuing task or an unCHECKed I/O is outstanding. The IOSW of the unsolicited interrupt will be moved to the 8-byte area specified in the input parameter list.

(7) Waits for telecommunications reception or transmission. Also waits for unsolicited interrupt from telecommunications device. If the TC I/O completes with an error because of missing device microcode or missing peripheral processor microcode, the error is logged but the microcode is not loaded.

Sequencing rules for alternation of RECEIVE/TRANSMIT followed by CHECK TCIO are enforced by the XIO SVC routine. XIO also checks that the device and DLP are not exclusively reserved by another task, and that the device is currently open (IPOPENed by this task).

CHECK TCIO may be issued without any previous I/O being issued provided the device is reserved by the calling task. In this case the CHECK is taken to be for an unsolicited interrupt from the DLP on the specified device. To receive an unsolicited interupt from the DLP, at least one of the devices on the DLP must have been IPOPENed and reserved by the caller. For this option, an IOSWADDR must be provided for the transfer of the IOSW to the caller.

If an unsolicited IOSW was returned by the DLP and the user has issued an XIO and is awaiting the completion IOSW to that I/O, the unsolicited IOSW does not cancel the effects of that condition. That is, the user is able to receive the unsolicited IOSW, and allowed to reissue the CHECK TCIO to receive the IOSW in response to the XIO. The general status byte of the IOSW returned will indicate to the user that it is an unsolicited IOSW rather than a normal IOSW in response to an XIO. If the user has received an unsolicited IOSW while waiting for the completion of an outstanding XIO, he must wait for the completion of the XIO by resubmitting the CHECK TCIO before issuing another XIO on the specified VS/DLP I/O channel.

The issuing task will be cancelled if the device is not reserved for use by the issuing task or an unCHECKed I/O operation is outstanding.

Read Volume Table Of Contents Block

READVTOC (SVC 19) NON-RESIDENT

Inputs: The top word of the system stack addresses the following argument list:

Input Argument List	preceding data	
	not used	size depends on options
	directory name/ OFB ptr (First 4 bytes)/not used	8 bytes
	starting item number = n >= 0	2 bytes
	count number = m >= 0	2 bytes
	not used	1 byte
	option number = 0, 1, 2, 3, or 4	1 byte
	volume name	6 bytes

Option number = 0 - To read VTOC attributes:

i. VTOC extents in use; no. of unused blocks in VTOC. 255 is returned if the number of unused blocks is greater than or equal to 255.

ii. Total no. of directories on volume; total no. of files on volume.

iii. Total no. of free extents on volume; total size of free extents.

iv. m largest free extents on volume.

Option number = 1 - To list m free extents starting from nth (from 1) free extent in VTOC.

Option number = 2 - To list m directory names and corresponding no. of files in directory starting from nth directory name (from 1) in VTOC.

Option number = 3 - To list m file names starting from nth file (from 1) in specified directory.

Option number = 4 - To read in m consecutive VTOC Control blocks starting from nth block in VTOC and copy into a file specified by the given OFB pointer.

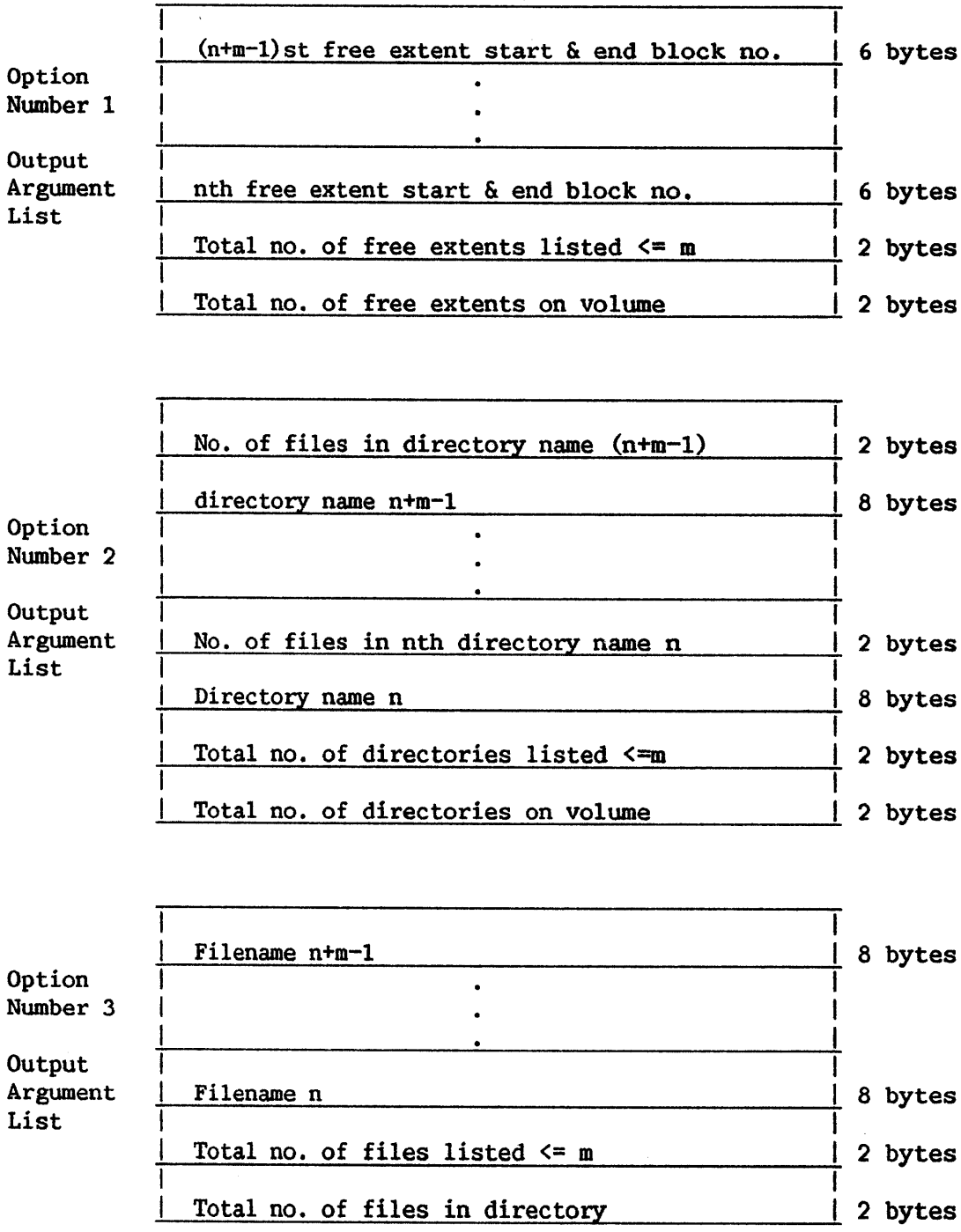
Outputs: A return code on top of system stack replacing inputs.

Return code = 0 - Requested function performed.
Return code = 4 - Invalid argument list address.
Return code = 8 - Volume not mounted.
Return code = 12 - Volume used exclusively by other user.
Return code = 16 - All buffers in use.
Return code = 20 - Invalid option request or OFB address.
Return code = 24 - Directory not found.
Return code = 28 - VTOC error; FDX1 and FDX2 do not agree.
Return code = 32 - Disk I/O error; VTOC unreliable.

When return code = 0, the input argument list is replaced by one of the following output argument lists depending on the option specified.

Option Number 0 Output Argument List

mth largest free extent start & end block number.	6 bytes
. . .	
1st largest free extent start & end block number.	6 bytes
Total size of free extents.	4 bytes
Total no. of free extents.	4 bytes
Total no. of files on volume.	2 bytes
Total no. of directories on volume.	2 bytes
3rd VTOC extent start & end block numbers.	6 bytes
2nd VTOC extent start & end block numbers.	6 bytes
1st VTOC extent start & end block numbers.	6 bytes
Number of VTOC extents in use.	1 byte
Number of unused blocks in VTOC.	1 byte


Option Number 1
Output Argument List
(n+m-1)st free extent start & end block no.
6 bytes
nth free extent start & end block no.
6 bytes
Total no. of free extents listed <= m
2 bytes
Total no. of free extents on volume
2 bytes
Option Number 2
Output Argument List
No. of files in directory name (n+m-1)
2 bytes
directory name n+m-1
8 bytes
No. of files in nth directory name n
2 bytes
Directory name n
8 bytes
Total no. of directories listed <=m
2 bytes
Total no. of directories on volume
2 bytes
Option Number 3
Output Argument List
Filename n+m-1
8 bytes
Filename n
8 bytes
Total no. of files listed <= m
2 bytes
Total no. of files in directory
2 bytes

Option Number 4 Output Argument List	not used	16 bytes
	# of blocks read (<m)	2 bytes
	Total VTOC size in blocks	2 bytes

Additional Output for option number 4: nth through (n+m-1)st VTOC control blocks copied to the file specified by the given OFB.

Function: To read the specified information from VTOC of a specified volume.

Note: The size of the input argument list must be big enough to hold the desired output argument list for successful operation.

Request Parameters

GETPARM (SVC 20) NON-RESIDENT

Inputs: Parameter list of eight or twelve bytes on stack top:

(1) One-byte Request Type Indicator:

Bits 0-3: Header Type/Acceptable Response Designator

0 = Request for Information. Acceptable response is modification of variable fields with completion signaled by pressing the ENTER key or any enabled PROGRAM FUNCTION KEY. By default, all PFKs are disabled.

1 = Request for Selection. Acceptable response is selection indicated and signaled by pressing the ENTER key or any enabled PROGRAM FUNCTION KEY. By default, all PFKs are enabled.

3 = System Request for Information. Reserved for use by OPEN and SYSTEM INITIALIZATION. The generated display header is different from that for type 0. Acceptable response is the same as that for type 0.

4 = System Request for Device Action. Reserved for use by OPEN, MOUNT, and CHECK. Acceptable response involves performance of machine operator duties with completion signaled by an interrupt from the indicated device, or by pressing the ENTER or any enabled PROGRAM FUNCTION KEY. By default, only PFK #16 is enabled, and by convention, it is used to allow the user to request termination of the device action request.

Bit 4: If set, indicates that the ENTER key will be accepted as a response to the GETPARM (ENTER key disabled), in addition to any keys specified in the PF key mask. This bit is ignored unless bit 5 is set.

Bit 5: PFK Mask Present Indicator.

0 = Default Mode. PFK's are enabled or disabled according to default values. The PFK mask should not be present in the parameter list.

1 = Override Mode. PFK's are disabled as indicated by the PFK mask which must be present in the parameter list.

Bit 6: Request Sequence Identifier

0 = TYPE I - Initial request for specification of information, selection among alternatives, or response.

1 = TYPE R - For correction of information, selection, or response just received as a result of the previous request.

Bit 7: User-Interaction Suppressor

0 = Normal Mode

(This mode will generate a workstation interaction even if the default data in the receiving fields of all Field Format Control Blocks satisfy lexical requirements for correctness. The workstation interaction is suppressed only if the procedure-specified data supplied is lexically correct and is sufficient in conjunction with the default data to completely satisfy the request.)

1 = Default Mode

(This mode is intended for use by OPEN only. This mode accepts procedure-supplied data if available, but will not generate a workstation interaction unless a field default value is lexically in error.)

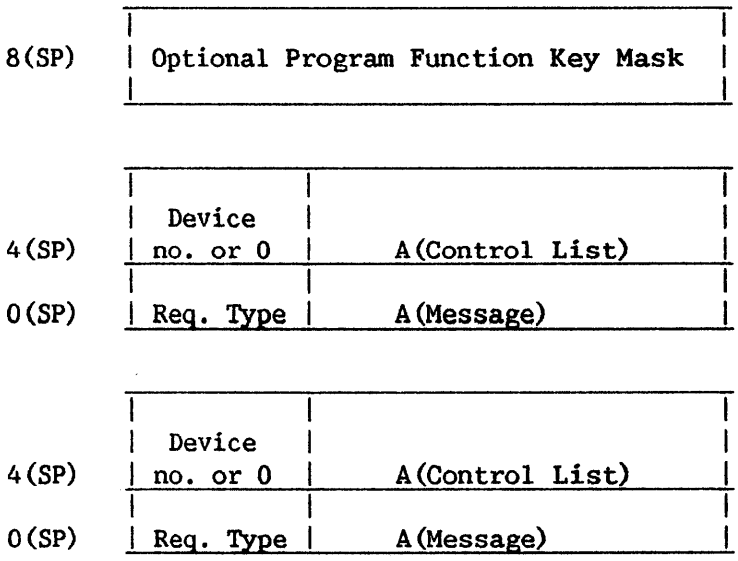
(2) Three-byte address of message to be displayed. (See format.)

(3) One-byte zero, or for Device Action request only, the device number of the device requiring service.

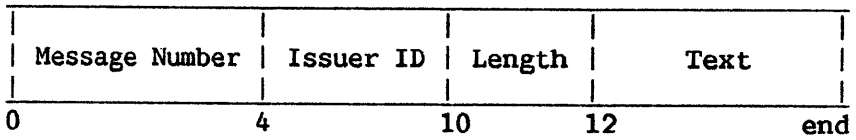
(4) Three-byte address of Parameter Group Control List. (See format.)

(5) Optional four-byte Program Function Key Mask. Each bit indicates whether the corresponding PFK should be enabled or not. The high-order bit corresponds to PFK.

1 = PFK enabled
0 = PFK disabled



The message at the specified address is in the following format:



The fields in this format are:

(1) Four-byte message number in ASCII characters. This number is displayed in the GETPARM header on any associated screen transactions.

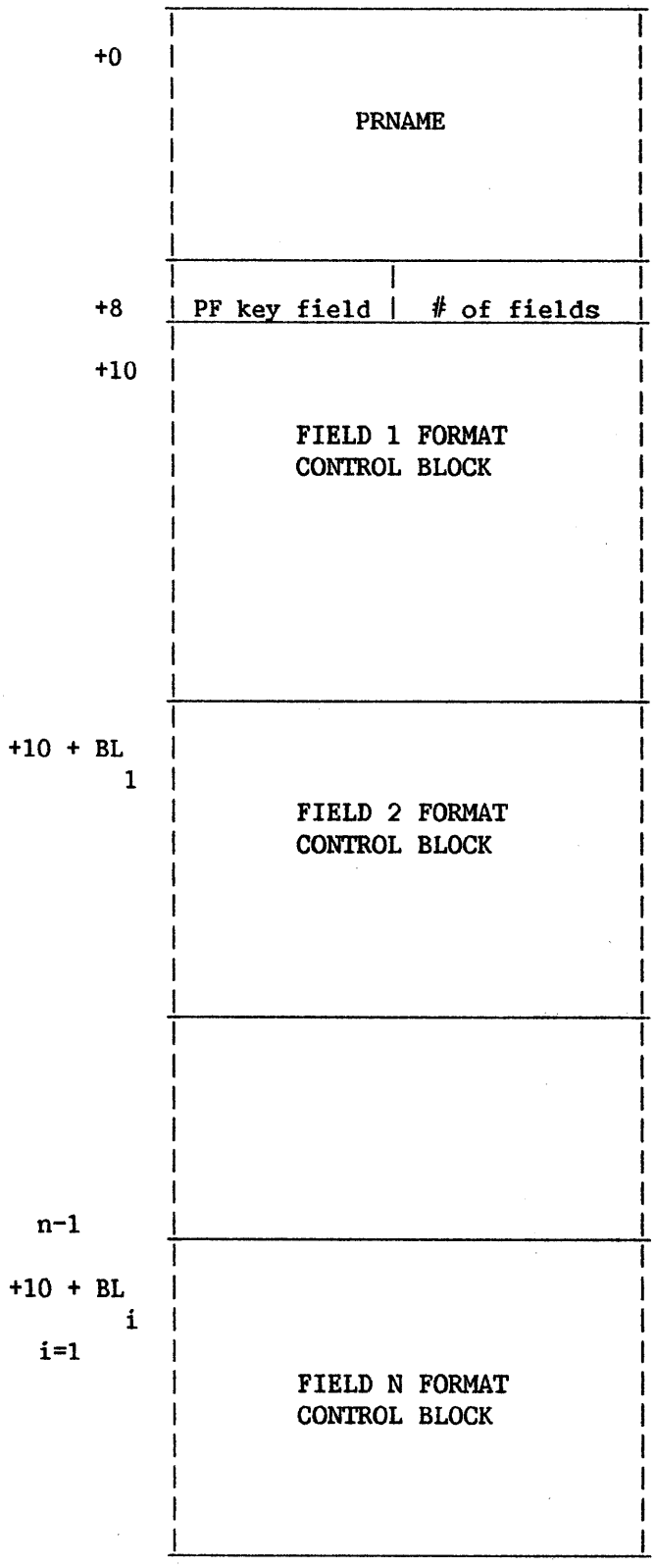
(2) Six-byte issuer ID in ASCII characters. This ID is displayed in the GETPARM header on any associated screen transactions.

(3) Two-byte message length in binary. This is the length of the text which follows.

(4) Message text in ASCII characters. If the message is longer than 79 characters, an end-of-line can be indicated by an ASCII "new-line" character.

No line may be longer than 79 characters excluding the end-of-line indicator. The message text is displayed beginning in column 2 of line 7. Each new line begins in column 2 of the next line. Lines longer than 79 characters are truncated. The last line does not require an end-of-line indicator.

The Parameter Group Control List is in the following format:



The fields in this format are:

(1) Eight-character, left-justified Parameter Reference Name (i.e., PRNAME).

(2) One-byte receiving field for AID character of program function key response. (This field may be set by procedure specification of a function key number.)

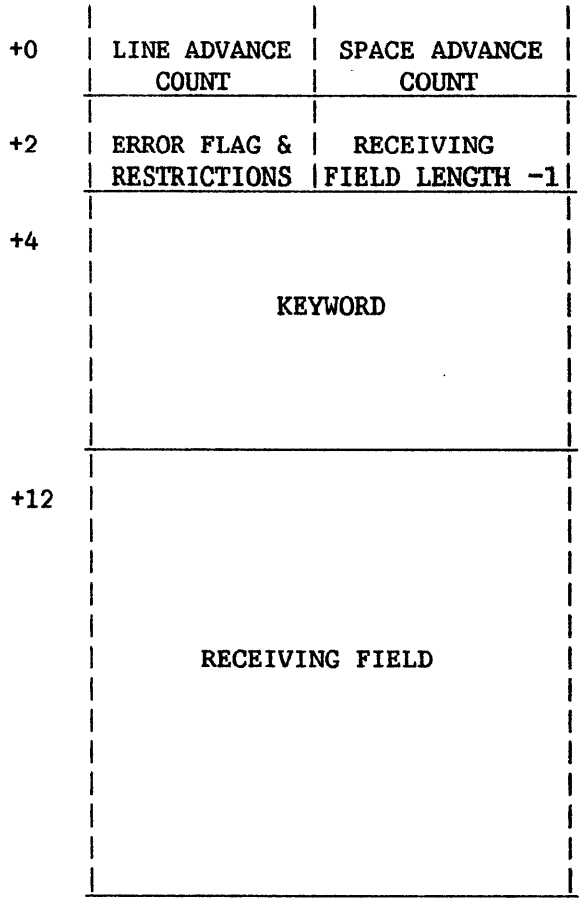
(3) One-byte binary count - number of Field Format Control Blocks.

The following is repeated for each displayable field.

(4) ... (n)

Variable length Field (display and entry) Format Control Block. There are two formats for the Field Format Blocks: one for control of the keyword/receiving field pairs, and the other to control the use of displayable imbedded text.

The Field Format Control Blocks for the keyword/receiving field pairs are in the following format:



The fields in this format are:

(1) One-byte binary Line-Advance-Count for display control.

(2) One-byte binary Space-Advance-Count for display control. (Line advance takes place before space advance. Both take place before display of keyword and receiving field.)

(3) One-byte binary Field Error Flag and Receiving Field Entry Restriction Indicator:

Bit 0: Field Error Flag (1 = error) (Set by program to draw attention to fields in error - Reset by GETPARM.)

Bits 5-7: Entry restrictions

0 = Character-string

No restrictions on content; maximum usable field length is 68 characters.

1 = Positive integer

Nonblank response need not be justified, but must consist entirely of the numerals 0-9 with leading and trailing blanks ignored. All blanks will be treated as a legitimate NULL specification. Field length is restricted to 16 characters.

2 = Numeric

Response must consist entirely of the numerals 0-9 optionally containing one decimal point and optionally preceded by a + or -. Leading and trailing blanks will be ignored. All blank response will be treated as a legitimate NULL response. Field length is restricted to 16 characters.

4 = Uppercase alphanumeric

All entered letters will be converted to uppercase. A legal nonblank response must be left-justified and consist entirely of the numerals 0-9, the letters A-Z, the national characters (@, #, or $), and trailing blanks. An all blank response will be treated as a legal NULL response indicator. Maximum usable field length is 68 characters.

5 = Uppercase hexadecimal

All entered letters will be converted to uppercase. A legal nonblank response need not be justified, but must consist entirely of the numerals 0-9, and the letters A-F with leading and trailing blanks ignored. All blanks will be treated as a legitimate NULL specification. Maximum usable field length is 68 characters.

6 = Uppercase Character String

All letters are converted on entry to uppercase; maximum usable field length is 68 characters.

7 = Alphanumeric Limited

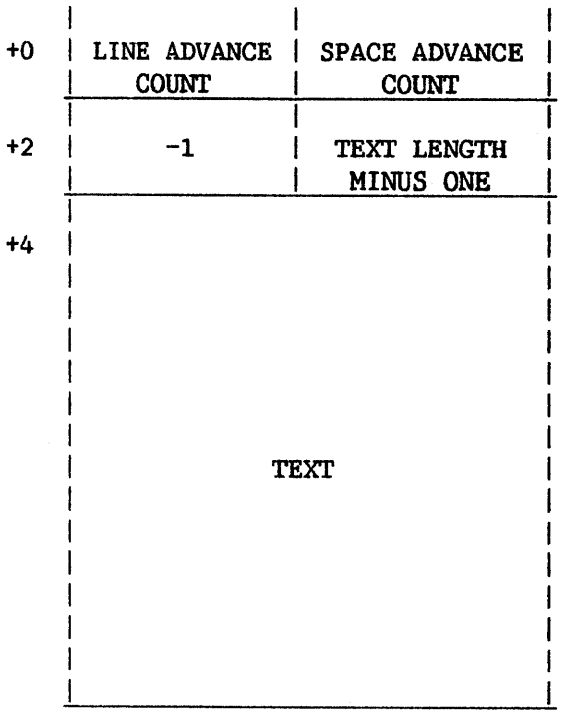
All entered letters will be converted to uppercase. Legal nonblank response will be left-justified, begin with a letter A-Z, or one of the national characters (@, #, or $), and consist entirely of the numerals 0-9, the letters A-Z, the national characters, and trailing blanks. An all blank response will be treated as a legal NULL response indicator. Maximum usable field length is 68 characters.

(4) One-byte binary receiving field length minus one (in characters).

(5) Eight-character, left-justified keyword used for display purposes (and to support noninteractive access via the Procedure Interpreter).

(6) Variable-length receiving field with default or current value in place.

The Field Format Control Blocks for imbedded text are in the following format:



The parameters in this format are:

(1) One-byte binary Line-Advance-Count for display control.

(2) One-byte binary Space-Advance-Count for display control. (Line advance takes place before space advance. Both take place before display of keyword and receiving field.)

(3) -1 (=255).

(4) One-byte binary text field length minus one (in characters).

(5) Variable-length text field.

Outputs: (1) Receiving fields as modified by user interaction or procedure specified data.

(2) Program Function Key Receiving Field set to accepted AID byte or procedure specified value.

(3) Field error flags in control list reset to ZERO.

(4) Input parameters popped from stack upon return.

Function: The GETPARM SVC enables user programs (and OPEN) to solicit and accept run-time parameter information, and to display and wait for acknowledgment of run-time messages. The requested information (or response) is gathered either through direct interaction with the user (at the workstation) or by calling prespecified data in the procedure which invoked the program. The program issuing the GETPARM SVC need not be aware of the data source; any interactive program that communicates with the user exclusively using GETPARM requests which can be identified and anticipated, can also be run in batch mode from a procedure. The parameters supplied to GETPARM are primarily related to the generation of a meaningful display.

When displayed at the workstation, the GETPARM request generates a header section, followed by the program-supplied message, followed by the keyword identified receiving fields and imbedded text section. These requests are divided into three functional types. A request for information should be indicated whenever one or more receiving fields are present. The expected user response is to modify one, all, or none of these fields and to signal when ready by pressing the ENTER key.

A request for selection should be indicated when a list of valid choices has been displayed (rather than modifiable receiving fields) and the expected user response is to identify and signal his choice by pressing one of the program function keys or the ENTER key. A request for acknowledgment should be indicated when the user has been asked to take some operator

action, or merely acknowledge receipt of an informational message. In this mode, the expected user response is to press the ENTER key as a ready signal.

When the issuance of a GETPARM SVC results in a screen display, the contents of the screen (if in use) are saved and are restored when the user indicates completion of his response.

GETPARM parameters should always be encoded with the assumption that they must be capable of generating an acceptable display. Lines 1 through 6 of the displays are reserved for system-generated headers which assist the user in responding to the GETPARM request. The headers are varied according to request type with special handling given when the issuer is OPEN. These lines include display of the message number and the issuer ID in the fixed format section of the message. The message text is placed on the screen beginning with column 2 of line 7. One additional line beginning at column 2 is displayed for each line-feed encountered. The line-feed is not displayed and does not use a character position. The maximum length for a line of text is 79 characters. The message section is completed with a blank line.

The receiving field display section begins with column 2 of the next line after the message section. Each receiving field is displayed with its associated keyword as follows:

(a) The screen line position advances the indicated number of lines from the current position. If line advancement takes place, column position is set to 2.

(b) The screen column position advances the indicated number of spaces from the current position.

(c) The eight-character keyword, followed by a blank, followed by an "=" character, followed by a blank, followed by the receiving field, followed by a blank is displayed. If any part of the receiving field is not on the screen, the keyword and field will not be displayed and will not be validated. Fields flagged as being in error will be blinked to attract user attention.

Each imbedded text field is displayed as follows:

(a) The screen line position advances the indicated number of lines from the current position. If line advancement takes place, column position is set to 2.

(b) The screen column position advances the indicated number of spaces from the current position.

(c) The variable length text is displayed followed by a blank. If any part of the text is not on the screen, no text is displayed.

The message text plus the receiving field display will be truncated if it exceeds 18 lines of displayable information.

Default or current information in the receiving fields is displayed as is without regard for entry control information. However, this information will be flagged for user correction if the format does not match when the user presses the ENTER key.

After the display is generated, the cursor is placed at the beginning of the first receiving field, or on type "R" request, to the beginning of the first keyword field which has an error flag set. A read is then issued to wait for a legal user response. Upon a signal from the user, the receiving fields are checked for legal contents and, if all are correct, are moved into the program's receiving fields. If any are in error, an error message is generated in the display header, the field in error is blinked, and another read is issued. When the user has successfully supplied the information of his choosing, control is returned to the user's program, and if the screen was in use, its contents are restored.

The user can suspend GETPARM processing and enter the Help Processor by pressing the HELP key. The screen will be saved as is, and the issuance of a CONTINUE command will return control to GETPARM with a restored screen. The user can then reenter his program by completing his response to GETPARM.

The program using GETPARM cannot assume that the user's response (or the accessed procedure data) will be valid. Consequently, the displayable parameters presented to GETPARM include a parameter-group receiving section with imbedded explanatory text, and a separate message section. The GETPARM user is expected to initiate a sequence of repeated requests until an acceptable response is received. During this sequence the requested information should be the same, while the message changes to best explain the difficulties encountered in the previous responses.

The request sequence indicator is used by GETPARM to differentiate between an initial request and a request for correction of material which the program did not find satisfactory. An initial request (Type "I") will be satisfied using procedure-specified data (located using the keywords and the parameter reference name) if available, generating a user-interaction only if all

such data has been exhausted. (Each initial request with a given PRNAME will consume one equivalent specification statement in a procedure.) A request for respecification (Type "R") will always generate a user-interaction.

If user-interaction suppressor bit is set, initial requests will access procedure-specified data if available, but will not ordinarily generate a user-interaction.

Read File Descriptor Record

READFDR (SVC 24) NONRESIDENT

Inputs: Two words on the top of the stack, as follows:

Bytes 0-3: Address of the parameter list, in the format given below.
Bytes 4-7: Ignored on input but used for output.

Parameter List Format:

Bytes 0-7:	Primary search library name (LIBRARY)
Bytes 8-15:	File name (FILE)
Bytes 16-21:	Volume name for primary search library (VOLUME)
Byte 22:	Option flags:
bits 0-3	Reserved
bit 4 = 1	Alternate search library and volume (last two entries in parameter list) are present.
bit 5 =1	Read both the FDR1 and the first FDR2 (if any), in ascending order, into the 160-byte area specified by the FDR receiving area. The FDRn field is ignored.
bit 6 =1	Read the file prologue along with other options that are set.
bit 7 =1	Read the file prologue only.
Byte 23:	FDR record number: FDRn = 0 if 80 bytes of FDR1 are to be read; = 1 if 80 bytes of first FDR2 are to be read; = 2 if 80 bytes of second FDR2 are to be read, etc.
Bytes 24-27:	Address of FDR receiving area (AREA)
Bytes 28-31:	Address into which to read the file prologue. (Used only when bit 6 or 7 is set in option flag.)
Bytes 32-35:	Reserved; should be zeros.
Bytes 36-43:	Alternate search library name (ALTLIB)
Bytes 44-49:	Volume name for alternate search library (ALTVOL)

Note: The parameter list for READFDR may not be located in segment 1 (user program reentrant segment).

Outputs: A return code in the top word of the stack replacing inputs:

Return code = 0 - File label copied into memory
Return code = 4 - Volume not mounted
Return code = 8 - Volume exclusively used by other user, no read
Return code = 12 - All buffers in use, no read
Return code = 16 - Library not found
Return code = 20 - File label not found
Return code = 24 - Attempt to read a file prologue when none was present.
Return code = 28 - Unused
Return code = 32 - VTOC error. FDX1 and FDX2 do not agree
Return code = 36 - VTOC error. FDX2 and FDR do not agree
Return code = 40 - Invalid input parameters
Return code = 44 - Disk I/O error. Volume Table of Contents unreliable

If return code=0 only, the next word on the stack contains the disk address of the FDR record read, in the following format:

Byte 0 (high-order) - Record on block, from 0.
Bytes 1-3 - Block on volume, from 0.

If return code nonzero, the contents of this word are irrelevant on output.

When the alternate library name is supplied, the library name and volume name entries in the parameter list are modified if required to indicate the library in which the specified file was found. The alternate library is searched after the normal (Filename1) library.

Function: To locate a disk file in the Volume Table of Contents of the specified volume and copy its label (File Descriptor Record) into the specified 80-byte memory area.

Also to read the file prologue (only supported for Word Processing files) and return the prologue to the caller in the specified area.

Rename Disk File

RENAME (SVC 26) NONRESIDENT

Inputs: The top word of the stack addresses an argument list of the following format:

Bytes 0-7:	Old library name
Bytes 8-15:	Old file name or ignored
Bytes 16-23:	New library name (if only library is to be renamed) or new file name. (For "Full RENAME," the new file name.)
Bytes 24-29:	Volume name
Byte 30:	Option flag
bit 0 =1	Bypass expiration date check
bit 1 =1	Rename a library (Bytes 16-23 of the parameter list then contain a new library name).
bit 2 =1	File access rights for this request to be limited to user LOGON rights.
bit 3 =0	Reserved; must be zero.
bit 4 =1	"Full RENAME" (i.e., rename both file and library -- Bytes 32-39 must contain the new library name.
bit 5-7	Reserved; must be zero.
Byte 31:	Not used.
Bytes 32-39:	New library name (for "Full RENAME").

Outputs: The file or library identified by the old name is renamed (old file name is ignored for rename of just a library). Consequently the contents of the file name or library name entries of the file index records and/or file descriptor record in the Volume Table of Contents are replaced by the new file name and/or library name.

A return code is placed in the top word of the stack replacing inputs:

Return code = 0 -	File or library renamed.
Return code = 4 -	Volume not mounted.
Return code = 8 -	Volume used exclusively by other user.
Return code = 12 -	All buffers in use, no rename.
Return code = 16 -	Library not found.
Return code = 20 -	File not found.
Return code = 24 -	Update access to some fileprotection class denied, no rename.
Return code = 28 -	Unexpired file, no rename.
Return code = 32 -	File in use, no rename.
Return code = 36 -	VTOC error. FDX1 and FDX2 do not agree.
Return code = 40 -	VTOC error. FDX2 and FDR do not agree.
Return code = 44 -	Invalid argument list address.

Return code = 48 - I/O error. Volume Table of Contents unreliable.

Return code = 52 - New file name or library name already exists, no rename.

Return code = 56 - New file name invalid (or first character '#'), no rename.

Return code = 60 - The VTOC is currently full -- insufficient space exists for the new FDX1/FDX2 ("Full RENAME" only).

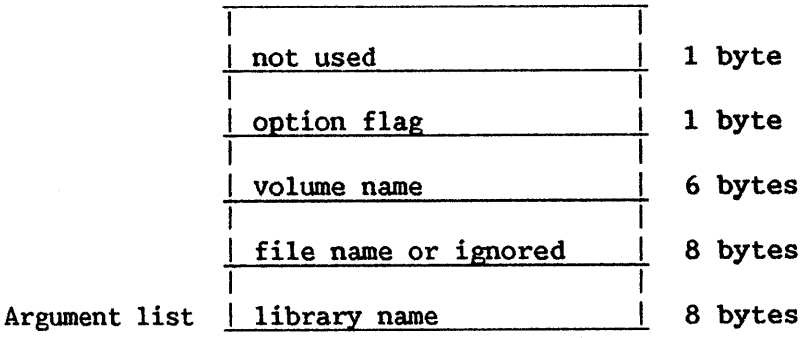
Return code = 64 - Reserved bits in the Parameter list Options Byte are non-zero.

Function: To change the name of a disk file or library on a volume. The structure of the Volume Table of Contents may be affected in the case of a "Full RENAME." The file must not be in use (open) when the RENAME is attempted, or it will fail (with return code 32). Similarly, no file in a library to be renamed may be in a file protection class for which the issuer does not have update access rights. Likewise, no file in such a library may be unexpired unless option flag bit 0 is set.

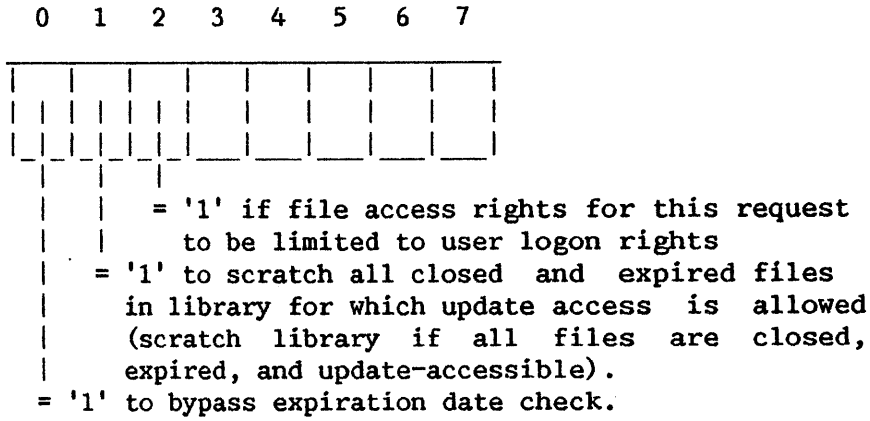
Scratch Disk File

SCRATCH (SVC 27) NONRESIDENT

Inputs: The top word of the stack addresses an argument list of the following format:



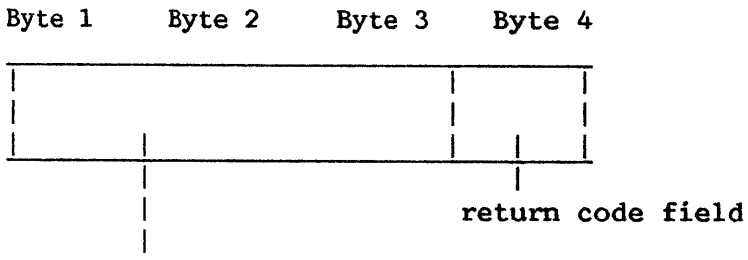
The option flag indicates whether password checking is to be bypassed:



File name: When specified, indicates the specific member in the library to be deleted. Ignored if option bit 1 (scratch entire library) is set.

Outputs: The file descriptor record of the file specified by the given filenames is scratched from the Volume Table of Contents of the specified volume and the disk space occupied by the file is deallocated. If an entire library is to be scratched, the file descriptors for all included files are eliminated and all space is deallocated.

A two-field return code on top of the stack replacing input:



Lost extent size in blocks during scratch file or library (size equals zero for no extent lost)

Return code = 0 - File or library scratched from volume
Return code = 4 - Volume not mounted
Return code = 8 - Volume used exclusively by other user
Return code = 12 - All buffers in use, no scratch
Return code = 16 - Library not found
Return code = 20 - File not found
Return code = 24 - Update access to file protection class denied (singlefile scratch only)
Return code = 28 - Unexpired File, no scratch (singlefile scratch only)
Return code = 32 - File in use, no scratch
Return code = 36 - VTOC error. FDX1 and FDX2 do not agree
Return code = 40 - VTOC error. FDX2 and FDR do not agree
Return code = 44 - Invalid argument list address
Return code = 48 - I/O error. Volume table of contents unreliable
Return code = 52 - Open, protected, and/or unexpired file(s) bypassed in scratching library

Function: To delete a disk file or library from a volume.

Note: Scratching the only file in a library (directory) eliminates the library itself.

Extract Data From System Control Blocks

EXTRACT (SVC 28) NONRESIDENT

Inputs: A variable-length parameter list on the stack top. The high-order byte of the lowest-addressed word designates the class of data required.

Class codes are:

Binary 0	- Limited output
Binary 1	- Full output
Binary 2	- Program exception PCW
Binary 3	- User-supplied list of items requested
Binary 4	- User-supplied list of items requested with additional input required
Binary 5 to 255	- Reserved

For classes 0, 1, and 2, a one-word parameter is supplied. The high-order byte of the word designates the class code (i.e., binary 1, 2, and 3, respectively). The three low-order bytes contain the address of a segment 2 area to receive the data (which must be on the existing stack or in the I/O buffer area).

For class 3, the parameter list consists of a header word and a list of 8-byte entries corresponding to the requested entries. The high-order byte of the header word designates the class code (i.e., binary 3), the next lower byte is reserved and should be zero, the two low-order bytes contain a count of the item entries that follow. Each 8-byte entry is in the following format:

Bytes 0-3 - Item identifier code. See EXTRD macro for a list of possible codes.
Bytes 4-7 - Address of a segment 2 area to receive the item.

For class 4, the parameter list consists of a header word and a list of 8-byte entries corresponding to the requested entries. The high-order byte of the header word designates the class code (i.e., binary 4), the next lower byte is reserved and should be zero, the two low-order bytes contain a count of item entries that follow. Each 12-byte entry is in the following format:

Bytes 0-3 - Item identifier code. See EXTRD macro for a list of possible codes.
Byte 4 - Length in bytes of area to receive the data.
Bytes 5-7 - Address of a segment 2 area to receive the data.
Bytes 8-11- Address of additional input for this item. See EXTRD macro for a description of these items.

Outputs: For classes 0-2, data in area addressed by word on stack, as follows, by ascending addresses:

Class 0:

(1) Total physical area in bytes not currently resident (4 bytes).
(2) Number of files which current task may open simultaneously (2 bytes).
(3) Workstation number associated with requesting task, or -1 if none (2 bytes).
(4) Remaining stack space in bytes after return from 'EXTRACT' (4 bytes).

Class 1:

(1) Total physical area in bytes not currently resident (4 bytes).
(2) Number of files which current task may open simultaneously (2 bytes).
(3) Workstation number associated with requesting task, or -1 if none (2 bytes).
(4) Remaining stack space in bytes after return from 'EXTRACT' (4 bytes).
(5) One day in clock units (4 bytes).
(6) System default library's volume name (6 bytes).
(7) System default library name (8 bytes).
(8) Task's default printer number, or -1 if none (2 bytes).
(9) User program library volume (6 bytes).
(10) User program library name (8 bytes).
(11) Current file-access bit map for 'execute' access from Program File Block (PFB--4 bytes).
(12) Default nonoutput volume for 'OPEN' (6 bytes).
(13) Default nonoutput library name (8 bytes).
(14) Current file-access bit map for 'read' access from Program File Block (4 bytes).
(15) Default output volume for 'OPEN' (6 bytes).
(16) Default output library name (8 bytes).
(17) Current file-access bit map for 'update' access from Program File Block (4 bytes).
(18) Number of segment 2 buffer pages currently available (2 bytes).
(19) Print output mode (1 byte).
(20) Default output file-access protection class, or blank (1 byte).
(21) User logon identification (3 bytes).
(22) Paging priority from TCBSCC (1 byte).
(23) Suggested lines-per-page for print files (1 byte).
(24) Operating System version number (A packed number VVRRPP, where 'VV' is the version, 'RR' is the revision, and 'PP' is the patch level) (3 bytes).

Class 2:

(1) Program Control Word (PCW) at time of most recent program exception for which a user exit was specified (8 bytes).

Class 3:

Data is returned as specified in the parameter list. (See EXTRACT macroinstruction description for possible values.)

Class 4:

ID=EXTRDIDDEVICE:

(1) Device class (1 byte).
(2) Device type (1 byte).
(3) Usage - 'EX' (exclusive), 'SH' (shared), or 'DT' (detached) (2 bytes).
(4) Task identifier of device owner, or -1 if none (4 bytes).
(5) Volume name of removable volume (disk or tape only). Blank if nothing mounted. (6 bytes).
(6) Volume name of fixed volume (disk only). Blank if nothing mounted (6 bytes).
(7) 4 bytes of binary zeros (reserved).

ID=EXTRDIDVOLUME:

(1) Device address, or -1 if volume not mounted (1 byte).
(2) Volume type: 'F' for fixed, 'R' for removable, or blank if not mounted. (2 bytes).
(3) Label type: 'SL' (standard label), 'NL' (no label), or blank if not mounted. (2 bytes).
(4) Usage--'SH' (shared), 'RR' (restricted removal), 'PR' (protected), 'EX' (exclusive), or blank if not mounted..
(5) Task identifier of volume mounter, or -1 if none (4 bytes).
(6) Blocks per cylinder (2 bytes).
(7) Maximum transfer in bytes (2 bytes).
(8) Cylinders per volume (2 bytes).
(9) Cylinders per physical volume, including bad or unused blocks (2 bytes).
(10) Number of files open on this volume (2 bytes).
(11) Sector type (diskette only): soft sector (S), hard sector (H)
(12) Addressing in effect (diskette only): Non-standard (N), Standard (S)
(13) Unused (2 bytes)

ID=EXTRDIDOTASK:

(1) Workstation device number of task specified, or -1 if none (1 byte).
(2) Current user ID for task specified, or blank if none (3 bytes).
(3) Current user name for task specified, or blank if none (24 bytes).
(4) Type ('F', 'FS', 'B', 'BS') of task specified (see TASKTYPE) (2 bytes).
(5) 18 bytes of binary zeros (reserved).

ID=EXTRDIDTAPEVOL:

(1) Device address, or -1 if volume not mounted (1 byte).
(2) 1 byte of binary zero (reserved).
(3) Density, BPI in binary: 556, 800 or 1600 (2 bytes).
(4) Label type: 'AL' (ANSI), 'NL' (no label), 'IL' (IBM label), or blank if volume not mounted. (2 bytes).
(5) Usage--'SH' (shared), 'EX' (exclusive), or blank if not mounted. (2 bytes).
(6) Task identifier of tape mounter, or -1 if none (4 bytes).
(7) Current file sequence number (2 bytes).
(8) Six bytes of binary zeros (reserved).

ID=EXTRDIDDEVLIST:

(1) Total number of devices for specified device class (1 byte).
(2) Number of device addresses supplied (1 byte).
(3) Device address list (1 byte for each device address).

ID=EXTRDIDDLPNAME:

(1) Bit map of devices on DLP (4 bytes)
(2) First device on DLP (2 bytes)
(3) Type of DLP (1 = 22V06-1, 2 = 22V06-2, 3 = 22V06-3) (1 byte)
(4) Number of lines (RS-232) controllable by the DLP (1 byte)
(5) Microcode file status (X'00' if stopped, X'80' if loaded) (1 byte)
(6) Reserved for future use (3 bytes)
(7) Microcode file name (8 bytes, zero if not loaded)
(8) Microcode library name (8 bytes, zero if not laoded)

(9) Microcode volume name (6 bytes, zero if not loaded)
(10) Reservation status of DLP (X'80' if reserved, X'00' if not reserved)
(11) Task number of the task which reserved the DLP (3 bytes)

ID=EXTRDIDDLPDEV#:

(1) Device status flag (X'80' if open, X'40' if reserved, zero otherwise)
(2) Task number of the task which reserved the DLP, or zero if device is unreserved (3 bytes)
(3) Name of the DLP on which the device is SYSGENed (4 bytes)

ID=EXTRDIDCLUSTER:

(1) Device number of the archiver driver, or zero (zero may indicate that there are no other devices on the cluster, or that among the devices there is no archiver drive)
(2) Unused (14 bytes)

ID=EXTRDVOLVCB:

(1) Volume Control Block address (4 bytes)

Function: Extracts data from system control blocks which may be of interest to user programs.

Mount Disk Or Tape Volume

MOUNT (SVC 30) NONRESIDENT

Input: The input list is 8 bytes long, unless the high-order bit of the first byte in the Volume Name field (Bytes 2-7) is set to 1, in which case the parameter list is 16 bytes long. The parameter list on top of the stack contains:

Byte 0 - Flags

For disk mount, the flags are used as follows:

```
Bit 0 = 1 - To mount an unlabelled volume.
      = 0 - To mount a standard labelled volume.
Bits 1-2 -  Two bits used for volume usage description:
      =00 - Mount for shared use.
      =01 - Mount with restricted removal.
      =10 - Mount with protected use, and restricted
            removal.
      =11 - Mount for exclusive use.
Bit 3 = 1 - Mount a fixed volume.
      = 0 - Mount a removable volume.
Bit 4 = 1 - 'No message' option (see function below).
Bit 5 = 1 - Mount volume for bypass-label-processing
            (see function below).
Bit 6 = 1 - The volume to be mounted allows spool files.
Bit 7 = 1 - The volume to be mounted allows work files.
```

For tape mount, the flags are used as follows:

```
Bit 0 = 1 - To mount an unlabelled volume.
      = 0 - To mount a standard labelled volume.
Bit 1 = 1 - Volume mounted for exclusive use.
      = 0 - Volume mounted for shared use.
Bit 2 = 1 - Unused.
Bit 3 = 1 - Mount an IBM tape volume.
      = 0 - Mount an ANSI tape volume.
Bit 4 = 1 - 'No message' option (see function below).
Bit 5 = 1 - Mount volume for bypass-label-processing
            (see function below).
Bit 6-7   - (Unused).
```

Byte 1 - Device number in binary (0-255).

Bytes 2-7 - Volume name (if high-order bit is set to 1, then an 8-byte extension is added to the parameter list).

Parameter List Extension

Byte 8 Bit 0 = 1 - Non-standard addressing in effect (for soft-sectored diskettes only).
Bit 1 = 1 - 'No display option': do not display message on user's workstation.
Bits 2-7 - (Unused - must be zero)
Bytes 9-15 - (Unused - must be zero)

Outputs: Binary Return Code in the top word of the stack, replaces the input parameter.

The following return codes are set when the new volume is physically mounted on the drive, and the corresponding VCB updated with the mounter TCB address, new volume label type, volume name and VTOC Block address (for labelled volume only). The sharing status ('shared' or 'exclusive') and the initialization status are updated as specified in the 'MOUNT' input parameter.

0 - Success.
4 - Successful mount, but new volume label type does not agree with input parameters
8 - Successful mount, but new volume name is not the volume name requested
12 - Disk or tape I/O error detected while reading the new volume label or the new volume has a bad VTOC. VCBSER is set to blank. This return code is set when the new volume is physically mounted on the drive, but the VCB cannot be filled in.

The following return codes are set without the mount message being shown on the workstation. The VCB for the volume is unchanged by the mount attempt.

16 - Device is not a disk or a tape, or device number is invalid.
20 - Device is detached.
24 - Disk does not have the requested volume type (fixed or removable).
28 - Request to mount an unlabelled volume on a disk unit other than an 2270V diskette.
32 - Input volume name is blank.
36 - Requested volume is already mounted on a disk unit. Also set for a duplicate volume name.
40 - Volume currently in use (by the operating system or user).
44 - Currently mounted volume reserved by another user for exclusive use.
48 - I/O buffer space insufficient to perform mount.
52 - Cannot allocate space for Tape I/O control blocks.

56 - Invalid request: work and/or spool filing requested in a non-labelled volume.
60 - Invalid request: non-standard addressing attempted with standard label option or on a hard-sectored device.
64 - Wrong media: soft-sectored diskette inserted into a device for hard-sectored diskettes only.
68 - Wrong media: hard-sectored diskette inserted into a device for soft-sectored diskettes only.
72 - Wrong media: hard-sectored diskette inserted for a non-standard addressing request.
76 - Wrong addressing mode: MOUNT request is for standard addressing but diskette is non-standard.
80 - Device reserved by another user.
84 MOUNT failed: aborted by user or operator request.

Function:

To mount a disk or tape volume. The input parameters are first validated, and if successful, a mount message will be displayed on the workstation to direct the user to mount the proper volume, unless the NODISPLAY option is chosen, in which case the message will appear only on the operator console. When the volume is mounted and the device is ready, the new volume label will then be read and checked, and the information in VCB is updated.

The NO MESSAGE option indicates that the volume to be mounted is already on the drive. No mount message will be displayed, and the VCB information is updated from that volume label.

The BYPASS-LABEL-PROCESSING option is used by the disk or tape initialization program and the floppy copy program (FLOPYDUP). At SVC EXIT, the VCB information is updated as follows:

VCBSER is set to input bytes 2-7.
VCB is set to be NOVTOC to allow non-labelled processing.
VCB is set for exclusive use by the user.
VCBFLAGSINIT = 1 to indicate that the VCB information could be inconsistent with the volume label.

NOTE:

A non-standard addressing option is now supported which allows the user to format a soft-sectored diskette in any combination of sector size and density. The use of this option is intended to be limited to specialized utilities. User programs which employ this option are responsible for performing direct and sequential I/O on a physical-sector basis. The user program must calcu- late the sector size and addresses, set mode, and set density. When non-standard addressing is specified, the XIO SVC will not perform extent validation or address translation, but simply passes the address to the firmware via the I/O Control Word (IOCW).

Modify Program Exception Exit Status

PCEXIT (SVC 31) NONRESIDENT

Inputs: One or two words on stack top. The lower-addressed word contains class codes (0, 1 or 2 in binary) in its high-order byte. For class 0 only, the three low-order bytes of this word contain the specified exit address. The higher-addressed word, present for class 0 only, contains a bit map of exceptions for which the user exit is to be taken.

Class codes are:

0 - Establish new program exception exit address and conditions (exception list), saving the old status if any.
1 - Restore previous program exception status, discarding current status.
2 - Cancel all program exception exits for the current program, discarding all status.

The bit map of exceptions is as follows:

Bit 0 - Unused
1 - Operation
2 - Privileged Operation
3 - Execute
4 - Protection
5 - Addressing
6 - Specification
7 - Data
8 - Fixed Point Overflow
9 - Fixed Point Divide
10 - Decimal Overflow
11 - Decimal Divide
12 - Supervisor Call Range
13-15 - Unused
16 - Floating Point Overflow
17 - Floating Point Underflow
18 - Significance
19 - Floating Point Divide
20-23 - Unused
24 - Stack Overflow
25-31 - Unused

Outputs: Program exception element(s) chained or removed from chain rooted in TCBPXE of the issuer's Task Control Block. Input parameters removed from stack.

Function: To modify the handling of program exceptions occurring in unprivileged user code, supplying or eliminating a current user exit address to receive control in the event of such an exception.

Note: When a program issues a LINK supervisor call, any current user program exception exit is eliminated, but the current status is preserved for restoration by UNLINK. (Execution of PCEXIT SVC Class 1 will not restore a program check exit status existing prior to a LINK.)

Set Or Reset Timing Interval

SETIME/RESETIME (SVC 32) NONRESIDENT

Inputs: Word on top of stack. The high-order byte of the lower-addressed word contains a class code in binary as follows:

X'40' - Set timing interval to expire at time of day specified in the three low-order bytes of parameter word, where this time is in 1/100 seconds into a day, from midnight. To request expiration at some time tomorrow, the value supplied must be 24 hours plus the required time of day. A requested time less than the current time of day will result in immediate expiration.

X'00' - Set times interval to expire after the number of 1/100 second units in the three low-order bytes of parameter word have elapsed.

X'80' - Remove timing interval previously established (can require removal of a TQEL from the time queue).

Outputs: A Timing Queue Element (TQEL) is placed in time-sequenced order on the timing queue (from MCBTIMQ) or is removed from that queue. The clock comparator value is modified if required. The input word is removed from the stack. The issuing task continues.

Function: To establish a timer interval, which may then be awaited by means of the CHECK supervisor call, or to remove an unCHECKED interval.

PUTPARM (SVC 33) Non-Resident

Inputs: The top 28 bytes of the stack contain the parameter list as follows:

Byte 0: Flags

Bit 0 = 1		DISPLAY option
	= 0	ENTER option
Bit 1 = 1		REFER option
Bit 2 = 1		CLEANUP option
Bit 3 = 1		MERGE option
	= 0	NOMERGE option
Bit 4 = 1		Enable Repeat Count
Bit 5 = 1		CLEANUP labelled FMTLIST (Bit 2 must also be set)
Bits 6-7		Reserved, must be zero

Byte 1: The AID character of a PFkey to be passed to the GETPARM if this is a backward reference; otherwise, not used and must be zero.

Bytes 2-3: Repeat count as follows (if flag bit 4 is on):

X'0000'	Never repeat
X'1'-X'7FFF'	Repeat n times
X'8000'	Repeat indefinitely

If flag bit 4 is off, field is not used and must be zero.

Bytes 4-11: If REFER NOMERGE or labeled CLEANUP option, bytes 4-11 contain the label of the FMTLIST to be referenced. If REFERLABEL is used in PUT option, bytes 4-11 contain the REFERLABEL. For unlabelled CLEANUP option, bytes 4-11 are all blanks. Otherwise, byte 4 must be zero, bytes 5-7 contain the address of the supplied FMTLIST (the destination FMTLIST for the REFER MERGE option), and bytes 8-11 are unused and must be zero.

Bytes 12-19: The PRNAME to be associated with the FMTLIST.

Bytes 20-27: The label of the FMTLIST to be created or accessed. Blanks indicate that no label is specified. If the MERGE or MERGE,REMOVE option is being used, bytes 20-27 contain the label of the source FMTLIST.

Outputs: PUTPARM returns to the issuer eight bytes on the top of the stack:

Bytes 0-3: Return Code:

- 0 - Successful.
- 4 - Backward reference label not found.
- 8 - Bad FMTLIST supplied.
- 12 - Error found in previously constructed Parameter Reference Blocks (PRBs).
- 16 - Invalid input parameter while using CLEANUP option.
- 20 - Invalid input parameter while using MERGE option.

Bytes 4-7: If Return Code is 0, address of FMTLIST specified in the input or backward referenced.

Function: The PUTPARM SVC has three major functions. The primary function (PUT function) is to supply parameters to another program's GETPARMs before issuing the LINK SVC to invoke. The second function (the CLEANUP function) is to cleanup the various internal data structures created by the PUT function. The third function (the REFER function) is to allow the calling program access to any parameters which the user may have changed at GETPARM time (the MERGE option), or to return the address of a previously created and labelled FMTLIST (the NOMERGE option).

Both the PUTPARM macro and the LINKPARM macro call the PUTPARM SVC. The PUTPARM macro allows only the parameterization of another program (the PUT function), while the LINKPARM macro accesses all the functions of the PUTPARM SVC.

PUT PUTPARM's primary use is to enable a program to supply parameters to a GETPARM issued by another program (the PUT function). The program supplying the parameters must link to the program issuing the GETPARM via the LINK SVC. A program may not use PUTPARM to pass parameters to its own GETPARMs.

The parameters to be supplied to the GETPARM are contained in a format list (FMTLIST), created with the FMTLIST macroinstruction. (A FMTLIST is identical to a KEYLIST, except that a FMTLIST contains no PRNAME.) When a PUTPARM is issued, it verifies that the specified FMTLIST is in the proper format, then saves the FMTLIST in a segment 2 buffer for subsequent GETPARM use. PUTPARM also constructs a Parameter Reference Block (PRB) to save the label, PRNAME, display option, and certain other information. The PRB is constructed in the segment 2 buffer area allocated by the PUTPARM SVC and chained to the previously constructed PRBs.

When a GETPARM in the linked-to program is issued, it searches through the current link level's saved (and unused) PRBs for one whose PRNAME matches the PRNAME of the GETPARM's KEYLIST. If one is found, the value for the keywords in the FMTLIST will be copied to the GETPARM KEYLIST (left-aligned and truncated). To solicit modifications by the user, A GETPARM workstation transaction may be requested by selecting the DISPLAY option; otherwise, a workstation transaction is suppressed. The KEYLIST (possibly modified by the user) is merged back into the FMTLIST for later backward reference.

If more than one GETPARM is issued with the same PRNAME, the PUTPARM-saved FMTLISTs will be used in the order in which they were supplied to the PUTPARM SVC. Normally, no two GETPARM requests access the same FMTLIST. A FMTLIST may be declared to be for repeated use via the macro parameter REPEAT=.

A FMTLIST may be labeled (via the LABEL= parameter) for later use. The backward reference facility allows a program to reuse the (possibly updated) parameters of a labeled FMTLIST. If a backward reference label is supplied to the PUTPARM SVC rather than a FMTLIST (e.g., via the REFERLABEL= parameter of the LINKPARM macro), a pointer to the labeled FMTLIST will be stored thus causing GETPARM to reuse the labled FMTLIST.

As an example of the backward reference facility, suppose that the program being parameterized requests the same set of parameters several times and that the calling program is suppressing the workstation transactions. The calling program could issue LINKPARM PUT several times, each specifying fully the GETPARM parameters. If one of the parameters was in error, the user would be forced to correct each transaction. If instead, only the first LINKPARM PUT specified the parameters (and was labeled) and the others referred back to the first, the user would only have to correct the first transaction.

The PUTPARM SVC also supports an override facility. If the PRNAME specified by the linking program matches the LABEL of a FMTLIST specified by the linked-to program, the parameter values in the linking program's FMTLIST will override those of the linked-to program's FMTLIST. Parameters not specified by the linking program retain the values specified by the linked-to program.

For example, suppose program 1 issues the following LINKPARM (FMTL1 sets KEY2 to 'PROG1'):

```
LINKPARM PUT, PRNAME='OVERRIDE',FMTLIST=FMTL1
```

and then links to program 2. Now suppose that program 2 issues the following LINKPARM (FMTL2 sets KEY1 and KEY2 to 'PROG2'):

```
LINKPARM PUT,PRNAME='DEMO',LABEL='OVERRIDE',
      FMTLIST=FMTL2
```

and then links to program 3. A GETPARM for PRNAME 'DEMO' by program 3 will set KEY1 to 'PROG2' and KEY2 to 'PROG1'.

As well as passing parameters to GETPARMs, PUTPARM may also pass a PFkey. This may be done in one of two ways, via either the PFKEY= or AID= parameter. Both can pass the full range of 32 PFkeys plus ENTER. PFKEY= takes either the actual key number (1-32) or the keyword ENTER. AID= takes the 'AID' character of the PFkey, where 'A'-'P' correspond to PFkeys 1-16 respectively, 'a'-'p' correspond to PFkeys 17-32 respectively, and @ corresponds to the ENTER key. Both methods have the same result (PFKEY= values are translated into AID= values for the SVC by the macro). The way in which the PFkey is passed to GETPARM depends on whether the LINKPARM is a normal or a backward reference.

In the normal case, the PFkey is placed into the first byte of the FMTLIST addressed by FMTLIST= by the LINKPARM macro. Note that the original FMTLIST is modified. In the case of a backward reference, the PFkey is placed onto the stack and then into the FMTLIST buffer. The original FMTLIST is not modified in this case.

CLEANUP The CLEANUP option is used to deallocate all the PRBs (and their associated FMTLISTs) chained to the Program File Block (PFB) of the current link level and above. This option enables the user to free the segment 2 buffers allocated for PUTPARM use. If no REFERLABEL is provided on the call, all PRBs and FMTLISTs at the

current link level and above are removed. If a REFERLABEL is provided, only the PRB and associated FMTLIST referenced by REFERLABEL is removed. The CLEANUP option may be used concurrently with the REFER option via specification of the REFER,REMOVE option in the LINKPARM macro (see below). The CLEANUP function is useful for programs which loop executing a large number of LINKPARMs to prevent FMTLIST buffers from becoming full.

REFER,
NOMERGE — The REFER,NOMERGE function of the PUTPARM SVC is to return the address (in the segment 2 buffer) of a previously created and labeled FMTLIST without the overhead of creating a new FMTLIST or a reference pointer. This function is used primarily by the Procedure Interpreter.

REFER,
MERGE — This feature is used primarily by programs which desire to keep track of any GETPARM parameters which a user might have overridden. This option allows the user of the LINKPARM macro to specify both a FMTLIST and a REFERLABEL. The contents of the FMTLIST addressed by the REFERLABEL= (the source) are merged into the FMTLIST addressed by FMTLIST= (the destination). Fields which are present in the destination but not the source are left unchanged. Fields which are present in the source but not the destination are ignored. The MERGE option may be combined with the CLEANUP option (the MERGE option is performed first) via the REMOVE operand.

Set Task-Related Defaults

SET (SVC 35) NONRESIDENT

Inputs: A variable-length parameter list on the stack top, consisting entirely of address pointers or words containing binary zeros. The last word of the list must have its highest bit on ('1'). Nonzero words address data items which replace corresponding task defaults as follows:

Index of Parameter List Address	Default	Length of Item	Corresponding Procedure Keyword
0	ETCBLINKVOL	6	PROGVOL
4	ETCBLINKNAME	8	PROGLIB
8	unused	-	-
12	ETCBDEFVOL	6	INVOL
16	ETCBDEFFILE1	8	INLIB
20	unused	-	-
24	ETCBDEFVOL0	6	OUTVOL
28	ETCBDEFFILE10	8	OUTLIB
32	ETCBDEFVOLS	6	SPOOLVOL
36	ETCBDEFVOLW	6	WORKVOL
40	ETCBDEFPRT	1	PRINTER
44	ETCBPRTTYPE	1	PRNTMODE
48	ETCBFPCLASS	1	FILECLAS
52	ETCBLINEAGE	1	LINES
56	ETCBPRTCLASS	1	PRTCLAS
60	ETCBFORMNO	1	FORM#
64	ETCBUPDVOL	6	RUNVOL
68	ETCBUPDNAME	8	RUNLIB
72	ETCBJOBSTATUS	1	JOBQUEUE
76	ETCBJOBCLASS	1	JOBCLASS
80	ETCBJOBLIMIT	4	JOBLIMIT

Outputs: The task's Extended Task Control Block is modified as requested. The parameter list is removed from the stack.

Function: To allow programs (especially the system's Procedure Interpreter) to modify default values. Most of these can also be modified by means of the 'SET' command.

Fields ETCBLINKVOL or ETCBLINKNAME modified by means of this SVC routine will be restored to their previous values when the issuing program UNLINKs.

Transmit Intertask Message

XMIT SVC 36 RESIDENT

Inputs: Two words on top of the stack as follows:

Byte 0, Bit 0 =	0	Wait if not enough buffer space.
	1	NOWAIT option, return if not enough buffer space.
1 =	0	OTHERTASK option, transmit only to other tasks.
2-7 =	0	(Reserved)
Bytes 1-3		Address of a message to be transmitted.
Bytes 4-7		Name (any characters) of receipt port for messages.

Outputs: The supplied message is placed in a system message buffer. This is copied to the address specified by the receiver as a result of the CHECK SVC routine with the MESSAGE option. The first two bytes of the supplied message indicate its length, including those bytes, and must be not greater than 2016.

Return codes are placed in a word on the stack top, replacing the inputs:

0 - Successful.

4 - No receiving port with the specified name.

8 - Unable to insert message in receiving port's message buffer -- insufficient remaining space in message buffer (NOWAIT option only).

12 - Unable to insert message in receiving port's message buffer due to receiving port's use of PRIVILEGED option.

16 - Message not transmitted; OTHERTASK option was specified and the designated message port belongs to the XMIT-issuing task.

Function: To communicate between user tasks, or between a user task and a specific subsystem of the operating system.

Create Intertask Message

CREATE (SVC 37) NONRESIDENT

Inputs: Two words on top of stack, as follows.

Byte 0, Bit 0 =	0	Receive all messages.
	1	Privileged Option.
1-7 =	0	(Reserved)
Byte 1		Reserved (X'0').
Bytes 2-3		Space to be allocated in buffer to receive messages (not greater than 2016).
Bytes 4-7		Name (any characters) of receipt port for messages.

Outputs: Resident buffer with specified name created to receive intertask messages.

Return codes are placed in a word on the stack top, replacing the inputs.

0 - Successful.

4 - Another task has activated the specified port name.

8 - Same task has already activated the specified port name.

12 - GETMEM failure.

Function: Allows the issuing task to receive intertask messages sent by XMIT (SVC 36) to the specified port name, rejecting any messages from non-privileged state code or from tasks that are not dedicated system tasks if the port was created with the PRIVILEGED option.

Destroy Intertask Message Buffer

DESTROY (SVC 38) NONRESIDENT

Inputs: One word on top of stack, containing name of one of this issuer's message receipt ports.

Outputs: The message buffer associated with the specified name is eliminated. If no such message buffer has been CREATEd by this task, the issuer is informed by a return code.

Return codes are placed in a word on the stack top, replacing the inputs:

0 - Successful.

4 - One or more messages were not received and are lost; otherwise successful.

8 - No such message buffer was allocated by this task.

Function: To remove from the issuing task the ability to receive messages directed to the specified name.

Set Cancel Exit Options

CEXIT (SVC 39) NONRESIDENT

Inputs: One or three words on stack top. For the CANCEL option, a word of binary zeros is on the stack. Otherwise, three words of the following format:

Byte 0 Flags

Bit 0	=	1	SET Option.
Bits 1-2	=	00	Debug enabled.
	=	01	NODEBUG Option.
	=	11	DUMP option.
Bit 3	=	0	HELP Key enabled.
	=	1	HELP Key disabled.

Bytes 1-3 Address of User Cancel Exit Intercept Routine, or zero.

Bytes 4-7 Address of User-Supplied Message to be used in place of the "CANCEL PROCESSING" Menu descriptions, or zero.

Bytes 8-11 Reserved (must be zero).

Outputs: PFB updated with CEXIT options as specified with parameters. Input parameters removed from stack. Abnormal termination occurs if either an invalid Cancel Intercept Address is provided (Message 0001 by SVC39) or an invalid Cancel Menu Message Mask Address is provided (Message 0002 by SVC39).

Function: To set CEXIT options in the current link level PFB.

Dismount Disk or Tape Volume

DISMOUNT (SVC 41) NONRESIDENT

Inputs: 8 bytes on top of stack:

Byte 0

Bit 0 = 0 -- Dismount disk volume.
= 1 -- Dismount tape volume.
Bit 1 = 1 -- No display option: do not write to caller's workstation.
Bits 2-7 -- Reserved; must be zero.

Byte 1 -- Reserved; must be zero.

Bytes 2 - 7 -- Volume name.

Outputs: 4 bytes of Return Code on stack top, replacing input:

0 - Successful
4 - Input volume name is blank, or bytes 0-1 in input are nonzero
8 - Volume not found
12 - Volume not dismountable
16 - Device detached
20 - Volume in use by a user or the operating system
24 - Volume reserved by another user
28 - GETMEM failure
32 - Device is reserved by another task

Function: To perform disk and tape dismount operations for the volume specified in the input.

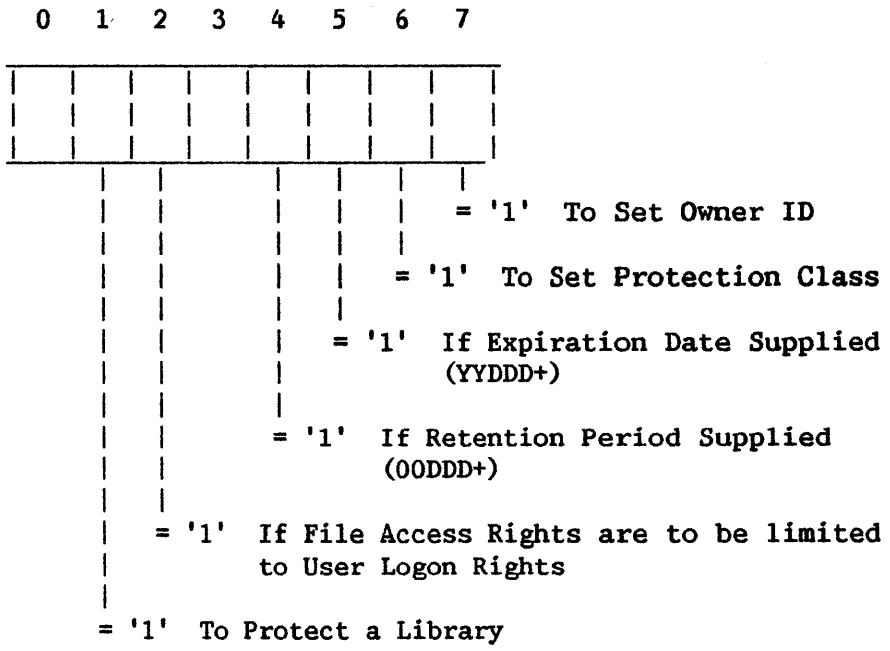
Protect File or Library

PROTECT (SVC 42) NONRESIDENT

Inputs: The top word of the stack addresses an argument list of the following format:

Argument List		
	New Expiration Date or Retention Period	3 Bytes
	New ID of Owner	3 Byte
	New Protection Class	1 Byte
	Option Flag	1 Byte
	Volume Serial	6 Bytes
	File Name or Ignored	8 Bytes
	Library Name	8 Bytes

The Option Flag indicates which protection data is to be changed and whether to protect a file or a full library:



Bits 0 and 3 should have values of 0.

Outputs: The protection attributes for the file or library identified is modified. Return codes in binary in the top word of the stack indicate the result of the request:

Return Code = 0 - Protection status successfully changed
Return Code = 4 - Volume not mounted
Return Code = 8 - Volume used exclusively by other user
Return Code = 12 - All buffers in use, no protection change
Return Code = 16 - Library not found
Return Code = 20 - File not found
Return Code = 24 - Update access denied, no protection change
Return Code = 28 - Unused
Return Code = 32 - File in use, no protection change
Return Code = 36 - VTOC error! FDX1 & FDX2 don't agree
Return Code = 40 - VTOC error! FDX2 & FDR don't agree
Return Code = 44 - Invalid argument list address
Return Code = 48 - I/O error! VTOC unreliable!
Return Code = 52 - Open or protected files bypassed in protecting library
Return Code = 56 - Invalid new protection data

Function: To update the protection information (protection class, owner of record, and/or expiration date) for a disk file or a library of disk files on a volume. The structure of the Volume Table of Contents (VTOC) is not affected by the change. No file that is to have its protection information modified may be open when the PROTECT is attempted.

Log Off Interactive Terminal

LOGOFF (SVC 43) NONRESIDENT

Inputs: Two words on stack top. These words are reserved for future use and currently must contain binary zeroes.

Outputs: PFB updated with the CEXIT option of NODEBUG and the logoff flag in the ETCB is set (ETCBFLGLOGOFF) for subsequent inspection by the appropriate Command Processor routines. A CANCEL SVC is issued with Message 0001 by SVC43. If the input parameter words are not binary zeroes, a CANCEL SVC is issued (Message 0002 by SVC43) and the logoff flag bit is not set.

Function: To effect logoff by program request.

Submit Job or Print Request

SUBMIT (SVC 46) NONRESIDENT

INPUTS: Accepts a parameter list of one word on the stack top:

Byte 0: Operation (Binary value, 1-255)

=1 Submit job for processing.
=2 Request printing of print file.

Bytes 1-3: Address of a 44-byte parameter list (word-aligned) for the operation.

Operation: Submit Job for Processing

Parameter List Format:

Bytes 0-7 Procedure name
8-15 Library name
16-21 Volume name
22-29 Job name or blanks (optional)
30 Job class (A-Z)
31 DUMP options:
X'80' User has specified DUMP or NODUMP on CANCEL (ETCBBGDUMPOPT)
X'40' Force DUMP on CANCEL (ETCBBGDUMP)
32-35 CPU time limit in timer units; if zero is supplied, then there is no time limit.
36 Initial status of job when queued:
X'80' Hold Not eligible for scheduling until released by the operator or submitter.
X'00' Active Eligible for scheduling upon submission of the request.
37 X'80' Test for time limit up -- must be set of a time limit is specified (TCBTIMLIMCHK).
X'40' Force CANCEL if limit is up (TCBTIMLIMCNCL).
X'20' Force Pause/HELP if limit is up (TCBTIMLIMPAUSE).

(If neither TCBTIMLIMCNCL nor TCBTIMLIMPAUSE is set and a CPU time limit is set, then a warning will be issued.)

X'04' Disposition after job processing: Requeue after execution.

38-43 Reserved (should be zeros).

Outputs: A return code is returned in the stack top word, with the following meaning:

- 0 - Successful
- 4 - Volume not mounted
- 8 - Volume in exclusive use
- 12 - All buffers in use -- unable to perform verification
- 16 - Library not found
- 20 - File not found
- 24 - Improper file type (or zero records as indicated in FDR1NRECS).
- 28 - File access denied
- 32 - VTOC error, FDX1 and FDX2 do not agree
- 36 - VTOC error, FDX2 and FDR do not agree
- 40 - Invalid specification of file, library, and volume
- 48 - System task not running, no spooled printing or interactive jobs
- 52 - Error in performing XMIT to System task
- 56 - Invalid options specified in parameter list

Operation: Request Printing of Print File

Parameter List Format:

Bytes	0-7	Print file name
	8-15	Library name
	16-21	Volume name
	22	Print class (A-Z)
	23	Form # in binary (0-255)
	24-25	Number of copies in binary
	26	Initial status of this file when queued: X'80' Hold Not eligible for print scheduling until released by the operator or the submitter. X'00' Spool Eligible for printing upon submission of the request.
	27	Disposition after printing: X'40' - Requeue after print X'20' - Save after printing X'02' - Collective print
	28-43	Reserved (should be zero)

Outputs: A return code is returned in the stack top word with the same meaning as above for the SUBMIT job processing function.

Allocate Heap Storage

GETHEAP (SVC 56) NONRESIDENT

Inputs: The top 16 bytes of the stack contain a parameter list as follows:

Byte 0: Option flags:

Bit 0 =1 SEARCH flag. Search backward for the subpool name specified in the 'POOLNAME' parameter of the GETHEAP macro (Bytes 8-15 of this input parameter list), starting from the link level specified in the 'LINKLEV' parameter of the GETHEAP macro (Byte 4 of this input parameter list) and going backwards until the subpool is found or all the link levels are exhausted.

=0 Search only at the link level specified in the 'LINKLEV' parameter of the GETHEAP macro (Byte 4 of this input parameter list).

Bit 1 =1 CREATE flag. Create a new subpool with the name specified in the 'POOLNAME' parameter of the GETHEAP macro (Bytes 8-15 of this input parameter list) and at the link level specified in the 'LINKLEV' parameter of the GETHEAP macro (Byte 4 of this input parameter list). A backward search is never initiated if the CREATE flag is set, and the SEARCH flag, if specified, is ignored.

Bit 2 =1 ALIGN flag. A 2K-aligned block is requested by the user. Useful only when a multiple of 2K bytes is requested; ignored otherwise.

Bits 3-7 Reserved; must be zero.

Bytes 1-3: SIZE Size of the block. All sizes are allowed, but they will be rounded up to their nearest 8-byte multiple.

Byte 4: LINKLEV Link level from which to start searching for the subpool. '0' means the current link level, '1' its parent, and so on. A value of 255 (X'FF') in this field represents the lowermost link level.

Bytes 5-7: Reserved; must be zero.

Bytes 8-15: POOLNAME 8-byte character string representing the subpool name. Blank names are not allowed. Trailing blanks are insignificant.

Outputs: A return code is returned in the stack top word. If Return Code = 0, the next word in the stack contains the starting address of the block; if Return Code = 4, the next word in the stack contains the size of the largest block available. Return codes have the following meanings:

0 - A buffer area has been allocated. The next word on the stack contains the block starting address. If requested, a subpool has also been created.

4 - Not enough space in segment 2. The next word on the stack contains the size of the largest block available. If requested, a subpool has also been created.

8 - Nonexistent link level specified.

12 - Nonexistent subpool name specified.

16 - User has overwritten area used by GETHEAP. User should CANCEL at this point.

20 - Error in parameter list. 'POOLNAME' all blank or a nonzero value in reserved fields.

24 - GETMEM failure. A new subpool cannot be created. This however does not prevent the user from allocating space from an existing subpool.

28 - CREATE failure. A subpool with the same name already exists at this link level.

Function: To allocate a block as requested. All block sizes including zero are legal. If, however, the block size is not a multiple of 8 bytes, the size is rounded up to the nearest 8-byte multiple. Maximum size is restricted only by the caller's segment 2 size less the size used by the system stack. The space is taken from the low end of segment 2. The value in control register 2 may be modified. Both the creation of a new subpool and allocation of a block out of the subpool can be accomplished in a single GETHEAP call. Successful creation of a subpool does not guarantee that a block of proper size can be allocated. There is no fixed space associated with the creation of a subpool; the space is allocated as and when requested.

Deallocate Heap Storage

FREEHEAP (SVC 57) NONRESIDENT

Inputs: The top 16 bytes of the stack contain a parameter list as follows:

Byte 0: Option flags:

Bit 0 =1 SEARCH flag. Search backward for the subpool name specified in the 'POOLNAME' parameter of the FREEHEAP macro (Bytes 8-15 of this input parameter list), starting from the link level specified in the 'LINKLEV' parameter of the FREEHEAP macro (Byte 4 of this input parameter list) and going backwards until the subpool is found or all the link levels are exhausted.

=0 Search only at the link level specified in the 'LINKLEV' parameter of the FREEHEAP macro (Byte 4 of this input parameter list).

Bit 1 =1 DELETE flag. Delete an entire subpool with the name specified in the 'POOLNAME' parameter of the FREEHEAP macro (Bytes 8-15 of this input parameter list) and at the link level specified in the 'LINKLEV' parameter of the FREEHEAP macro (Byte 4 of this input parameter list). A backward search is never initiated if the DELETE flag is set, and the SEARCH flag, if specified, is ignored.

Bits 2-7 Reserved; must be zero.

Bytes 1-3: SIZE Size of the block. All sizes are allowed, but they will be rounded up to their nearest 8-byte multiple.

Byte 4: LINKLEV Link level from which to start searching for the subpool. '0' means the current link level, '1' its parent, and so on. A value of 255 (X'FF') in this field represents the lowermost link level.

Bytes 5-7: BUFLOC Starting address of the block to be deleted.

Bytes 8-15: POOLNAME 8-byte character string representing the subpool name. Blank names are not allowed. Trailing blanks are insignificant.

When the deletion of an entire subpool is desired, the SIZE and BUFLOC parameters have no meaning and are ignored.

Outputs: A return code on the top word of the system stack with the following meaning:

0 - A buffer area has been deallocated or an entire subpool has been deleted.

4 - Invalid buffer address specified.

8 - Nonexistent link level specified.

12 - Nonexistent subpool name specified.

16 - User has overwritten area used by FREEHEAP. User should CANCEL at this point.

20 - Error in parameter list. 'POOLNAME' all blank or nonzero value in reserved fields.

Function: To de-allocate a block as requested. All block sizes, including zero, are legal, but they will be rounded up to their nearest 8-byte multiple. An entire subpool can also be deleted in a single FREEHEAP call, through the use of the DELETE flag. The value in control register 2 may be modified.

On UNLINK, all the subpools belonging to that link level are automatically deleted.

CHAPTER 7: DATA MANAGEMENT SYSTEM SERVICES

7.1 INTRODUCTION

The Data Management System (DMS) is described from the user program viewpoint in this section. The user program communicates with DMS routines through the User File Block (UFB). A user program has one UFB for each file it can process. When a UFB is connected to an OFB by SVC OPEN, the file is considered OPEN and may be processed through function requests to the Data Management System using the UFB. The system enforces a limit on the number of files a user program can have OPEN at any one time. A file is removed from the OPEN state by using SVC CLOSE. A user program's UFBs are located in the user's modifiable data area (Segment 2). SVC 0 (OPEN) and SVC 1 (CLOSE) are described in Chapter 6.

DMS function-routines are reentrant routines and are located as part of the system code (Segment 0). They execute in User State and use SVCs (XIO, CHECK, ALEX, UPDATFDR, DTI) to perform privileged I/O operations and to modify protected control blocks (OFB, FLUB, etc.). DMS function-routines can be used by multiple users; they perform as if they were an extension of the user program.

The UFB is logically divided into four sections as follows:

a. Access Method Section - This section contains some basic information used by the DMS function-routines.

   i. Five function vectors, corresponding to the five function-requests allowed on a file. A function vector consists of a modifier byte and the address of a DMS function-routine (loaded by OPEN).

   ii. Two user-supplied function-routine error-exit addresses.

   iii. The record-area pointer (user supplied).

   iv. The key-area pointer (user supplied).

v. Two File Status Bytes (UFBFS1 and UFBFS2) used to return an ASCII-character code to the user for each function-request.

b. File Location and Attributes Section - This section contains various information (assembled by SVC OPEN) about the file.

c. Data Management System Section - This section contains an Open File Block (OFB) pointer for the file, buffer-related information (including two Buffer control Blocks), and other data used by the Data Management System.

d. Indexed Disk File Extension - This Section is present for indexed disk files only; it contains information regarding the file index, etc.

7.2 VS DISK FILES

This section describes the DMS functions, open-modes, and access methods for disk files residing on a disk volume with a VTOC. DMS does not distinguish between disk volumes and diskette volumes.

a. A disk file resides on a disk volume. A disk file is fully contained within a disk volume (a file cannot span a volume).

b. Each file has an associated file label in the VTOC. File label format is described by the FDR1 and FDR2 control blocks.

c. The physical block size for a disk file is 2048 bytes. Records in a disk file are always blocked (in 2K physical blocks).

There are three access methods supported by the VS Data Management System:

1. Record Access Method (RAM).
2. Block Access Method (BAM).
3. Physical Access Method (PAM).

The VS DMS access methods are characterized by several factors such as:

a. unit of data transfer.
b. buffer support; blocking and deblocking support by DMS.
c. physical I/O support provided by DMS.
d. file-organization dependence.

The following pages within this section describe each of these access methods in detail for disk devices and disk files. Other files (on other devices) are also processed under one or more of the access methods. The other devices are described separately.

The pages at the end of this section (Notes on VS Disk Files) describe certain specific UFB and FDR (file label) fields that may be of interest to the assembly language programmer. Other topics such as NOVTOC diskettes are also noted.

7.2.1 Record Access Method (RAM) - Disk Files

The unit of data transfer is the logical record as indicated in UFBRECSIZE. This Access Method is the normal (default) access method assigned by SVC OPEN. Files are accessed in a file organization dependent manner. This access method includes:

a. random and sequential access of consecutive files (fixed length or variable length records).

b. sequential and keyed access of consecutive files (data records).

c. indexed file sequential creation.

d. sharing files for update.

Open Modes supported under RAM

a. Output mode - This mode supports the creation of a file. Records are presented by the user program in sequential order. For indexed files, each record must have a greater key value than the preceding record. The WRITE function request causes the record from the user-record-area to be the last record in the file.

The following modes are used with existing files.

b. Input mode - This mode supports the retrieval of records from a file. Records may be read (random, sequential or keyed) but may not be modified. Multiple user-programs can access the same file in Input mode since no modification is allowed. The READ function request causes a record from the file to be read into the user-record-area.

c. Input/Output mode (I/O mode) - This mode supports retrieval and modification of records in a file. Only one user-program (at a time) can access a particular file in I/O mode. The REWRITE function request is available to rewrite the (updated) record from the user-record-area to the file. The REWRITE function request accesses the last record read from the file by the program; REWRITE must be preceded by a READ(HOLD) function request. If the file is an indexed file, records may also be added (keyed, WRITE) or removed (DELETE) from the file in I/O mode.

d. Extend mode - This mode supports the addition of new records to the end of an existing disk file. Only one user program (at a time) can access a particular file in Extend mode. SVC OPEN positions the logical record pointer so that all WRITE function-requests add records to the end of the file. Extend mode is valid for consecutive files only (for indexed files, I/O mode provides the ability to add records to the files).

e. Shared mode - This mode supports all the functions supported in I/O mode except READ NODATA. However, shared mode also provides an update-interlock system which allows multiple user-programs to update the same file concurrently. DMS grants exclusive control of a record to a program to insure that concurrent file updates are processed correctly. DMS provides all shared file support.

FUNCTION REQUESTS PROVIDED FOR DISK FILES UNDER THE DMS RECORD ACCESS METHOD

FILE ORGANIZATION - CONSECUTIVE
RECORD FORMAT - FIXED LENGTH RECORDS

	Input	Output	I/O	Extend	Shared
Read	X		X		
Write		X		X	
Rewrite			X		
Start		X		X	
Delete					

FILE ORGANIZATION - INDEXED
RECORD FORMAT - FIXED LENGTH RECORDS

	Input	Output	I/O	Extend	Shared
Read	X		X		X
Write		X	X		X
Rewrite			X		X
Start	X		X		X
Delete			X		X

FILE ORGANIZATION - CONSECUTIVE
RECORD FORMAT - VARIABLE-LENGTH RECORDS*

	Input	Output	I/O	Extend	Shared
Read	X		X		
Write		X		X	X
Rewrite			X		
Start	X	X	X	X	
Delete					

* Shared mode is supported for consecutive log files. (A consecutive log file must have variable-length records.)

Consecutive Disk File
Fixed Length Records (Blocked)

Disk Block (2K)

Record 1	Record 2	Record 3	Record 4	///

The fixed-length records are blocked; block size is 2048. Each block in the file except (possibly) the last block contains the same number of records. Records do not span blocks. A 2K disk block contains unused space at the end if the record size is not an even divisor of 2K.

Record Format

Record length is fixed. The file label contains fields indicating the number of records in the file (FDR1NRECS) and the number of records in the last block of the file (FDR1EREC).

The length of the record is a fixed value (UFBRECSIZE). This value is set when the file is created (OUTPUT mode). This value is held in UFBRECSIZE and used for data transfer between the user-record-area and the file when an existing file is processed under RAM. Valid record length is 1-2048.

Indexed Disk File
Fixed Length Records (Blocked)

Disk Block (2K)

BL	Record 1	Record 2	Record 3	///	Data Level Chain

Record Format

	Key field	

fixed length record

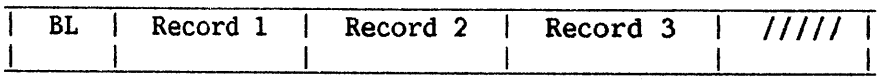
Records are fixed length. Each record has a key field upon which the records are ordered. The record length can range from 1-2040 bytes. The key length can range from 21-55 bytes. The key position can range from 1 up to any value such that the whole key fits within the length of the record. The key position field is numbered from 0 internally (UFBKEYPOS is numbered from 0).

The file label contains a record count (FDR1NRECS).

An indexed file contains both data blocks and index blocks. The index blocks form a tree structure by which data blocks may be located. Only DMS routines manipulate index blocks. Under RAM, the user program has access only to data records. Index blocks have a chain field and a block prefix (BL) similar to data blocks. The block prefix field (BL) is described in the next section on variable length records.

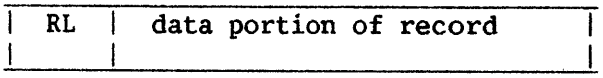
Consecutive Disk File
Variable-Length Records (Blocked)

Disk Block (2K)



The two-byte block prefix (BL) indicates the length of the block (i.e., sum of record lengths plus 2). DMS blocks records as tightly as possible during file creation (OUTPUT mode).

Record Format



The two-byte record prefix indicates the length of the record (i.e., data portion of record plus 2). UFBRECSIZE at SVC OPEN (OUTPUT mode) is the maximum record size; this value is stored in the label. Valid record length is 1-2024.

For OUTPUT mode, UFBRECSIZE indicates the length of the record in the user-record-area. The record prefix (RL) is not included in this length; the record prefix is not present in the user-record-area. UFBRECSIZE may be varied by the user program in order to write different sized records. The data management system supplies the BL and RL prefixes.

For existing files, UFBLRECSAVE equals the maximum record size (from the file label). UFBRECSIZE is set by DMS to equal the length of the record in the user-record-area after each read. The RL is not moved to the user-record-area (i.e., UFBRECSIZE is set to RL2). There are two minor restrictions for variable length consecutive files (as opposed to fixed lengths);

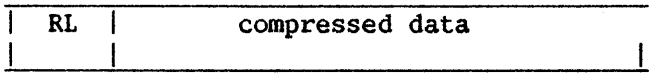
1. Read relative is an invalid function-request for variable length consecutive files. The START SKIP and START BEGIN function-requests are available for positioning within a variable length consecutive file.

2. The record size (UFBRECSIZE) may not be changed when REWRITING a record (flagged as invalid function-request).

Compression Option

A compression option is available for variable length consecutive files. This option can allow significant saving in disk space. The option is requested when the file is created. In this case, the data portion of the record is stored in compressed format (as described by the COMPRESS and EXPAND machine instructions).

Record Format



The use of the compress option is transparent to the user program under RAM*; however, the REWRITE function is not allowed for consecutive files with compressed variable-length records. DMS performs compression on OUTPUT (WRITE) and expansion on INPUT (READ). The value of UFBRECSIZE, UFBLRECSAVE and the file label is always the length of the noncompressed record (i.e., for a READ, UFBRECSIZE is set to the noncompressed length regardless of the value of RL). For compressed format, the maximum record size (established at file creation) is 2024.

NOTE:

Print records are stored as compressed variable length records when directed to a disk file.

* If Read Nodata is used, the user receives a pointer to the compressed data portion of the record (in a buffer). In this case, the user program must expand the record directly.

Indexed Files with Variable Length Records

The indexed filed organization is available with variable length records. In this case, UFBRECSIZE is used to indicate the length of the record read for existing files. The full range of function-requests available with indexed files (fixed-length records) is also available for indexed files with variable-length records. The compress option is also available. The length of the record may be changed when rewriting a variable-length record to an indexed file. DMS will handle any index updating required.

Function-Requests and Function-Request Modifiers - RAM

There are five DMS function-requests (READ, WRITE, REWRITE, DELETE, and START). Macroinstructions have been provided for these function-requests. The macroinstructions also provide for all the possible function-request modifiers. The preceding charts illustrate which of these function-requests are available under RAM for a particular Open Mode/Disk File Organization combination.

Read (NEXT, RELATIVE RECORD NUMBER or KEY) - The indicated record is read into a system buffer by DMS if not already present. The record is moved from the buffer to the user record area (addressed by UFBRECAREA). For fixed-length records, the record length is taken from UFBRECSIZE.

Read Modifiers

a. NEXT - The desired record is the next record in the file. After OPEN, READ NEXT yields record number 1.

b. REL - The desired record is indicated by the relative record number in the fullword addressed by the UFBKEYAREA. REL is only available for consecutive files with fixed length records.

c. KEYED - The desired record is indicated by the key starting at the location addressed by UFBKEYAREA. The length of the key is UFBKEYSIZE. KEYED is only available for indexed files.

d. NODATA - The record is not moved to UFBRECAREA; instead, the address of the record in the buffer is returned in general register 1.

e. HOLD - The HOLD modifier is used in I/O or Shared mode to indicate that the record just read may be rewritten or deleted. In this case, DMS will retain the current block for the record so that the rewrite or delete may occur. The HOLD modifier is required in all cases where the record is to be rewritten.

Read File Status With UFBEODAD Return

The following file status conditions cause a return to the program at the address in UFBEODAD. These conditions arise when the indicated record cannot be found.

FS='10' - End-of-data. A READ NEXT function request attempted to read past the end of the file.

FS='23' - Record not found. A READ REL function supplied a relative record number equal to zero or greater than the highest record number in the file; or a READ KEYED function-request supplied a key value which was not equal to any key value in the file.

Write

For OUTPUT mode (file creation) or EXTEND mode, the record is moved from the user-record area to the file. The record becomes the last record in the file. For indexed files, the key in the record written must be greater than the preceding key value.

For indexed files in I/O or SHARED mode, the record is moved from the user-record area to the file. The key field of the record must not be the same as a key already in the file. There are no modifiers for the write-function request.

Write file status with UFBEODAD return.

The following file status conditions may occur when using the WRITE function-request to process an indexed file under RAM.

FS='21' - Record key out of sequence - OUTPUT mode

FS='22' - Duplicate key - I/O or SHARED mode

FS='24' - End of primary extent--OUTPUT mode. Insufficient space was provided; the file may be CLOSED; additional records may be added in I/O mode.

Rewrite

The last record must have been read with the HOLD option. The record is moved from the user-record area to the file. For variable length records, the length must be unchanged. For indexed records, the key field of the record must be unchanged. There are no modifier values for rewrite.

Delete

The last record must have been read with the HOLD option. The record is removed from the file (indexed files only). The content of the user-record area is not used.

Start

The start function can be used for three different purposes under RAM:

a. Switch Processing Mode--this option is available for consecutive files, fixed length records, open mode of OUTPUT or EXTEND. The modifiers are START OUTPUT (sets the number of records in the file to zero), START EXTEND (allows the file to be extended by subsequent WRITES) or START I/O (allows READ and REWRITE operations after setting temporary end-of-file indicator); functions available are READ, REWRITE, and START OUTPUT and START EXTEND.

If Start I/O is used, an additional option is available. If UFBVAM is set in the modifier byte, then the Access Method will be switched. If switching from BAM to RAM, UFBNRECS may be provided to allow exact setting of the end-of-file indicator.

b. Position to a Record within the File--this is the common usage of START under RAM. Positioning within an indexed file is accomplished using modifiers EQUAL, GREATER THAN, or GREATER THAN OR EQUAL. Positioning within a variable-length consecutive file uses START SKIP where the word addressed by UFBKEYAREA indicates a signed number of records to be skipped. START BEGIN is also available to position to the first record of a consecutive variable-length file. The HOLD modifier is available for START on an indexed file. The generic keysize feature (UFBGKSIZE) is also available with the START request on an indexed file.

c. HOLD a file, or release a file or record from HOLD status (shared mode - ignored in IO mode).

START HOLD acquires protected update (HOLD) rights to the entire file. This HOLD status is released by START RELEASE, which removes any file or record from HOLD status for the issuer.

Start Function UFBEODAD Returns

These UFBEODAD returns may occur when START is used to position within a file.

FS='10' - end-of-file. End of file can occur when the value in UFBKEYAREA for START(SKIP) causes the end-of-file (positive value) or start-of-file (negative value) to be exceeded.

FS='23' - record not found when START EQUAL issued.

FS='24' - record not found when START (GREATER THAN) or START (GREATER THAN OR EQUAL) issued; key value is greater than any key in the file.

Notes for Record Access Method (RAM)

1. File size specification for file creation (OUTPUT MODE): The amount of disk space allocated for a file is determined by DMS. DMS uses a record number count supplied by the user program (UFBNRECS). For indexed files, DMS also calculates the additional space required for the index. Extra space can be released when the file is closed.

2. Buffers - The user program can specify the buffer size to be allocated under RAM. A large buffer size can represent a significant performance improvement for sequential access of the file. The buffer size is supplied in UFBBUFSIZE before SVC OPEN. The buffer size must be a multiple of 2K (otherwise the default size is used). The buffer size may also be adjusted by DMS depending on the maximum data transfer size supported by the device. Default buffer size is 2K.

3. Record size at SVC OPEN (NonOUTPUT mode) - For existing files, UFBRECSIZE is used to request a file with a specific record size. UFBLRECSAVE is filled in from the record size field in the file label. If UFBRECSIZE = 0 at SVC OPEN, the field is simply filled with the value from the file label.

4. Anticipatory buffer priming is automatically performed under RAM. Also, sequential rewrites (or deletes) are blocked.

7.2.2 Block Access Method (BAM) - Disk Files

The unit of data transfer is the physical disk block; the length of a disk block is always 2K (2048 bytes). This Access Method can be selected by setting UFBF1BAM before SVC OPEN. Files are accessed in a file-organization-independent manner. This access method supports:

a. random or sequential access of any disk file using relative block number in file (from 1). The unit of transfer is 2K bytes.

b. creation of new disk files by copying existing disk files in a file-organization-independent manner.

c. user program blocking or deblocking of records.

The Open Modes supported under RAM are also supported under BAM. The only significant difference is the unit of data transfer (always 2K under BAM).

FUNCTION REQUESTS PROVIDED FOR DISK FILES UNDER THE DMS BLOCK ACCESS METHOD

FILE ORGANIZATION - ANY
RECORD FORMAT - ANY

	Input	Output	I/O	*Extend	Shared
Read	X		X		
Write		X		X	
Rewrite			X		
Start		X		X	
Delete					

* - Extend Mode is not supported for indexed files under BAM.

Function-Requests and Function-Request Modifiers - BAM

The preceding chart indicates the function-requests available under BAM for a particular open mode. The Block Access Method allows the user program to access any disk file in a file organization independent manner.

Read - The indicated 2K disk block (NEXT or RELATIVE BLOCK NUMBER) is read into a system buffer by DMS if not already present. The block is moved from the buffer to the user-record area as addressed by UFBRECAREA; the length is 2048 bytes.

Read Modifiers

a. NEXT - The desired block is the next block in the file. After SVC OPEN, READ NEXT yields block number 1.

b. REL - The desired block is indicated by the relative block number in the full-word addressed by UFBKEYAREA (from one).

c. The NODATA and HOLD modifiers are available under BAM in the same way as described in RAM. (The KEYED modifier is not used under BAM.)

Read File Status Using UFBEODAD Return

The end of file (FS='10') and record not found (FS='23') conditions occur under BAM the same as they occur under RAM.

Write - The 2K block is moved from the user-record area to the file. The block becomes the last block in the file. There are no modifiers; the UFBEODAD return is not taken for a write function-request under BAM.

Rewrite - The last block must have been read with the HOLD option; i.e., the preceding function-request must have been a READ with the HOLD modifier. The block is moved from the user record area to the file. No modifiers are used with the rewrite command.

Start - The start function is used to switch processing modes in BAM. This function is available in OUTPUT or EXTEND modes. The modifiers allowed a START OUTPUT, START EXTEND and START I/O. No UFBEODAD returns are taken when switching modes.

Notes for Block Access Method (BAM)

1. File size specification for file creation (Output Mode): The amount of disk space to be allocated for a file may be specified using a record number count (from UFBNRECS) as in RAM. However, under BAM, the user program may optionally specify file size as a number of 2K blocks (UFBNBLKS) by setting UFBF4BLKAL. In this case, UFBNBLKS contains the number of blocks to be allocated; UFBNRECS is not used for disk space allocation. If UFBNBLKS is used and sufficient disk space is not available, the program will be cancelled (unless an OPEN EXIT has been supplied). The program will also be cancelled if UFBF4BLKAL is set and UFBNBLKS is zero.

2. Buffers - The user program can specify the buffer size to be allocated under BAM by supplying an appropriate value in UFBBUFSIZE before SVC OPEN. A larger buffer can improve performance for sequential access of the file.

3. Record Size at SVC OPEN - For Output mode, the value of UFBRECSIZE will be placed in the file label. For other modes, UFBRECSIZE may be used to specify the particular record size desired. After SVC OPEN, UFBLRECSAVE contains the record size (from the file label); UFBRECSIZE is set to 2048 after OPEN.

4. Anticipatory buffer priming is automatically performed under BAM.

5. Setting Number of Records in File at Close - The Block Access Method operates in a file organization independent manner. Therefore, at SVC CLOSE, additional information may be required in order to update the file label correctly for OUTPUT or EXTEND mode.

The number of the last block in the file is maintained by DMS in UFBEBLK. The additional information required at CLOSE is dependent on the file organization.

a. Fixed Length Consecutive - The total record count (UFBNRECS) should be supplied (this includes the number of records in the last (perhaps partial) block). The default is a DMS supplied count based on a full last block. The user supplied value (UFBNRECS) is ignored if it is inconsistent with UFBEBLK.

b. Variable Length Consecutive - The record count should be supplied (UFBNRECS). Otherwise, DMS will set a record count assuming that all records are maximum size. This approximate DMS default value is only used if no other value has been supplied (i.e., UFBNRECS=0).

c. Indexed Files - The following fields are required before SVC CLOSE: UFBEREC, UFBNRECS, UFBHXBLK, UFBPTRD, and UFBDABLK. These values should be taken from the Input file for the file copy application. (Other than the file copy application, it is not recommended that indexed files be created under BAM.)

7.2.3 Physical Access Method (PAM) - Disk Files

The unit of data transfer is defined by the user-program. No buffer support is provided by PAM. This Access Method can be selected by setting UFBF1PAM before SVC OPEN. Files are accessed in a file organization independent manner. This access method supports:

a. random and asynchronous access of any disk file.

b. use-initiated physical I/O operations of more than one physical disk block.

c. user-implemented buffering strategies.

The following Open Modes are supported under PAM:

a. Output mode - This mode supports the (sequential) creation of a file. The user-program controls the units of data transfer; the block number in file is adjusted by DMS for the length transferred. The data is written to the end of the file.

b. Input mode - This mode supports retrieval of data from the file. Data retrieval is random; the user-program supplies both the data length and the block number in file. The user-program may pass through the file sequentially by updating the block number appropriately.

c. I/O mode - This mode supports retrieval and modification of records in the file. The REWRITE function is random; the user-program supplies the data length and block number in file. In PAM, the REWRITE function need not be preceded by a read function.

FUNCTION-REQUESTS PROVIDED FOR DISK FILES UNDER THE DMS PHYSICAL ACCESS METHOD

FILE ORGANIZATION - ANY
RECORD FORMAT - ANY

	Input	Output	I/O	Extend	Shared
Read	X		X		
Write		X			
Rewrite			X		
Start	X	X	X		
Delete					

Function-Requests and Function-Request Modifiers - PAM

The function-requests used in the Physical Access Method have two common characteristics:

a. The data length desired is set in UFBBLKSIZE. UFBBLKSIZE is 2 bytes long; it is considered as a positive number. The data length specified must be a multiple of 2K. After the data transfer is initiated, UFBBLKSIZE is set to the data length actually transferred. The data length actually transferred may be less than the requested data length due to end-of-file truncation, end-of-extent truncation or end-of-cylinder truncation.

b. The user-program must issue a START(WAIT) function-request in order to wait for I/O completion. DMS does not wait for I/O completion in PAM.

Read - The data as indicated by the block number in the fullword addressed by UFBKEYAREA (block number from 0) and the contents of UFBBLKSIZE (data length) is read into the user-record-area addressed by UFBRECAREA. UFBBLKSIZE is adjusted, if necessary, to reflect the length of the actual data transferred. There are no modifiers. UFBEODAD return with FS='23' is taken if the starting block number is beyond the end of the file.

Write - The indicated data is written to the end of the file. UFBBLKSIZE is updated to indicate the length of the data transfer. DMS also updates the next block number for the data blocks transferred. After OPEN, the block number is 0. There are no modifiers.

Rewrite - The data as indicated by UFBRECAREA and UFBBLKSIZE is written to the file starting with the block number from the word addressed by UFBKEYAREA. UFBBLKSIZE is then adjusted to reflect the length of the data actually transferred. There are no modifiers. UFBEODAD return with FS='23' may be taken for Record Not Found (as in Read).

Start - The start function is primarily used with the WAIT modifier to wait for I/O completion for the preceding READ, WRITE or REWRITE command. In OUTPUT mode, the modifiers OUTPUT, EXTEND and I/O are available for mode switching (as in the other access methods).

Notes for Physical Access Method (PAM)

1. File size specification for file creation (Output mode). The amount of disk space to be allocated may be specified as in BAM (record count or block count). Record size at SVC OPEN is also handled as in BAM.

2. PAM uses no buffers, all I/O is performed using the user-record-area directly.

3. Setting the number of records in the file at SVC Close requires the same file-organization-dependent information as in BAM.

7.2.4 Notes on 2200VS Disk Files

1. File labels - The VTOC holds the file labels (File Descriptor Record - FDR) for all the files on a disk volume. The FDR fields defined below are of particular interest for DMS routines.

   a. FDR1RECSIZE - size of logical record. This field indicates logical record size for fixed-length records. For variable-length records, this field is the maximum record size for the file. This field is only set at file creation.

   b. FDR1BLKSIZE - blocksize; always 2048.

   c. FDR1NRECS - record count. This field indicates the number of logical records in the file. This field is not used by DMS as the end-of-file indicator.

   d. FDR1EBLK - last block in file. This field contains the block number (in file from 0) of the last used block in the file. This field is set for the last block number in file regardless of the file organization. This field (in conjunction with FDR1EREC if necessary) is used by DMS as the end of file indicator.

FDR1EBLK is a sufficient end-of-file indicator for variable-length records since in this case the end-of-block condition is determined by the contents of the block. Indexed files use the data chain for end-of-file detection; FDR1EBLK (plus 1) serves as a free space pointer in this case.

For fixed-length records, FDR1EREC is used as a count of the records in the last file block; i.e., for fixed-length records, the end-of-file indicator is FDR1EBLK with FDR1EREC.

FDR1EREC is 1 for variable-length records. For indexed files FDR1EREC is used as an index level count (used by DMS only).

FDR1NRECS, FDR1EREC an FDR1EBLK are always updated as a group by CLOSE. They are updated under RAM for OUTPUT and EXTEND modes and for indexed file SHARED and I/O modes.

FDR1RECSIZE is never changed by SVC CLOSE. DMS does not support the use of different logical record sizes under RAM. FDR1RECSIZE always indicates the noncompressed maximum record size for files containing compressed variable-length records.

2. UFB fields - The following fields are availble to the user-program for inspection during DMS processing. The uses of other UFB fields should not be assumed. For disk files, no fields other than the modifier bytes, UFBRECAREA, UFBKEYAREA, and the error addresses should never be changed by the user program while the file is open (except for setting required values in certain fields immediately before closing the file when using BAM or PAM). Also, UFBRECSIZE and UFBGKSIZE (indexed files) are modifiable in certain circumstances.

   a. UFBLF, UFBLFMOD - these fields contain the last function and last modifier value.

   b. UFBLRECSAVE - this field contains FDR1RECSITE after a file is OPENED.

   c. UFBRECSIZE - this field contains the record size for fixed-length records under RAM. This field contains the current record length under RAM for variable-length records. This field contains 2048 under BAM.

Note that UFBNRECS is not always maintained during DMS processing. Also, UFBNRECS is always set to 0 by SVC OPEN for OUTPUT mode. UFBNRECS is available after OPENING an existing file (equals FDR1NRECS). UFBRES3 contains the UFBNRECS value (used for space allocation) after SVC OPEN - OUTPUT MODE. (UFBRES3 may be re-used during DMS processing.)

3. Access Method notes - VS disk files can be accessed or created by any access method. (However, creation of an indexed file under BAM or PAM is generally restricted to the file copy case, since no index building support is supplied.) Selection of an Access Method will generally depend on the unit of data transfer desired or the file-organization support desired (e.g., indexed files under RAM). RAM provides the fullest DMS support. BAM can be used to advantage for consecutive files to reduce DMS overhead by performing user-program blocking and deblocking. Significant sequential performance improvements may be accomplished in RAM or BAM by requesting a larger buffer at OPEN. PAM offers the most flexibility and the least DMS support; it is recom- mended primarily when data movement is to be minimized or when a flexible user-supported buffering scheme is desired.

4. NOVTOC diskettes - This section has dealt with disk files which reside on a volume with a VTOC (i.e., disk files which have an associated disk file label). The diskette without a VTOC is used primarily as a medium for exchanging information between the VS system and other computer systems. The DMS support for NOVTOC diskettes includes all three access methods. In addition, the user-program can select any logical record size from 1-2048 since there is no file label information available. The following restrictions do notapply to NOVTOC volumes.

    1. indexed files are not supported
    2. extend mode and shared mode are not supported.

In order to have the full range of VS system support and utility support, it is recommended that NOVTOC volumes be copied to standard disk files before processing rather than being processed in place.

5. Specification of File Organization and Record Size at SVC OPEN.

    The fields UFBFORG and UFBRECSIZE may be supplied by the user program in order to request a specific file organization or record size. If these fields are zero, then any file organization or record size is allowed. These fields are both required (non-zero) for OUTPUT mode in which case the values are stored in the file label.

    The following file organizations are supported. (The only valid organizatons allowed in the file label.)

    a. Consecutive file; fixed-length records UFBFORGCONSEC.

    b. Program file; fixed-length records as described in VS Object format (1024 nominal record size) UFBFORGCONSEC + UFBFORGPROGRAM.

c. Consecutive file; variable length records UFBFORGCONSEC + UFBFORGVLEN.

d. Indexed files; fixed records UFBFORGINDEXED.

e. Indexed files; variable length records UFBFORGINDEXED + UFBFORGVLEN.

f. Print file; variable length records (compressed) UFBFORGCONSEC + UFBFORGVLEN + UFBFORGPRINT.

Notes on File Organization

a. For OUTPUT MODE, if UFBFLAGSCOMP is set (i.e., compression option), then SVC OPEN will force UFBFORGVLEN=1 so that a file with compressed variable length records will be created.

b. For nonOUTPUT mode, any consecutive file will be accepted if UFBFORG = UFBFORGCONSEC at SVC OPEN.

c. For any mode, if UFBFORG = UFBFORGCONSEC + UFBFORGPRINT, SVC OPEN will set UFBFORGVLEN and UFBFLAGSCOMP in order to generate a standard print file. (UFBFORGPRINT indicates that the user program has supplied a two-byte print control field for each record.)

Note

FDR1ORGCONSEC + FDR1ORGPRINT is not a valid value for the file label (FDR1ORG).

The following list indicates the valid range of values for UFBRECSIZE when a file is being created in OUTPUT mode. (These same values also apply when UFBRECSIZE is specified at SVC OPEN for nonOUTPUT mode.)

a. Consecutive file; fixed-length records 1-2048.

b. Indexed file; fixed-length records 1-2040.

c. Variable length records with or without the compress option - 1-2024 (represents noncompressed size).

The compress option may be set using UFBFLAGSCOMP when creating a file in OUTPUT mode. This flag is ignored for fixed length records. This flag is not considered to be part of the file organization.

7.3 2200VS INDEXED FILE SUPPORT

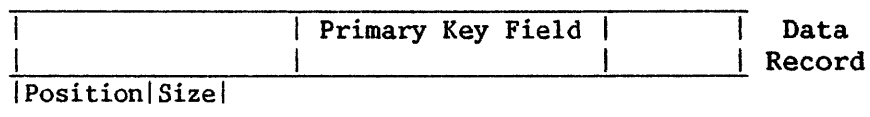
This section describes 2200VS Indexed Files and the access level support provided by the 2200VS Data Management System (DMS). Indexed files provide efficient sequential access and random access by primary key. The record formats available are fixed length, variable length, and compressed. The topics covered in this section are:

Indexed File Creation
Accessing an Existing Indexed File
Buffer Options for Indexed Files
Indexed File Structure
Functional Overview of Alternate Indexed File Support
Alternate File Error Log
Overview of Indexed and Alternate Indexed File Structures
Internal DMS Record Formats for Alternate Indexed Files
Shared Mode
Log Files
Advanced Sharing (Multiple Resources)
Detailed Functional Overview for Shared Mode
Summary of START Functions

This section describes file access using the Record Access Method (RAM). It is intended as a detailed guide to 2200VS DMS support of Indexed Files. This section also includes descriptions of actual 2200VS file structures. While an in-depth understanding of the file structure is not required to effectively access indexed files, the sophisticated programmer should be able to use the file structure information for performance and packing considerations and for lowlevel debugging.

Indexed File Creation

An Indexed file is created using OUTPUT mode. The number of records to be supplied in OUTPUT mode is used for initial file allocation. There is no restriction on the number of records which may be added later; also, no additional space for file expansion need be included at file creation. The format of a record in an indexed file is:



Primary key size and position is determined by the user. This information is stored in the file label. All records in the file must have a unique primary key value (duplicates not allowed). (For variable-length records, the minimum record size = position plus key size.) Primary key size can be from 1-255 bytes. Primary key position can be from 1 up to the maximum value (MAX = Record size minus key length). If data records are compressed, it is somewhat advantageous to have the Primary Key position be near the beginning of the record.

Records are added to the file in OUTPUT mode using the WRITE function. Each record must have a higher primary key value than the preceding record. The records are blocked and written to the file in OUTPUT mode. (The low-level index is also built as records are added; the full index structure is created only when the file is CLOSED (OUTPUT mode). This last feature (index creation at CLOSE) is used only in OUTPUT mode, i.e., in IO or SHARED mode the index structure is maintained for each function-request and no additional index update at all is required at CLOSE for these modes.

OUTPUT mode provides very fast sequential loading (creation) of an Indexed file. However, it is possible to create an indexed file containing no records (record count = 0); in this case, records could be later added randomly in IO or SHARED MODE. The WRITE function-request in OUTPUT mode is significantly faster than the dynamic WRITE request available in IO or SHARED modes.

A file packing option is available in OUTPUT mode. This option allows data blocks and index blocks to be filled to 100% capacity. The default is 100% packing. Loose packing may allow subsequent random record additions to be processed without requiring a block to be split. Loose packing also has a slight effect on efficient allocation of Data Base Key values (to be described later). It should be noted that loose packing is never required and that loose packing has no significant effect on the number of records which may be added to the file at a later time.

Accessing an Existing Indexed File

An existing Indexed file can be accessed in INPUT, IO or SHARED MODE. The functions available in INPUT MODE are:

READ NEXT (next in primary key sequence from current position)

READ KEY (primary key value supplied)

START (KEY) (modifiers and generic keysize options; START KEY used to establish file position for READ NEXT)

These functions allow records to be read; record modification is not allowed.

The additional functions available in IO MODE are:

REWRITE	must follow READ w/HOLD
DELETE	allows current record to be rewritten or deleted
WRITE	add record to file; record has unique primary key value.

The functions available in SHARED mode are the same as those available in IO mode.

The HOLD option for the READ function - READ NEXT and READ KEY may be used with the HOLD modifier (i.e., READNEXTHOLD and READKEYHOLD). The HOLD modifier bit has no effect in INPUT MODE. In IO mode the HOLD bit has two effects:

1. A REWRITE or a DELETE may be issued for this record.

2. The buffer containing the record will not be used for sequential buffer priming.

Buffer Options for Indexed Files

When an indexed file is OPENED buffers are assigned; these buffers are used by DMS when performing the user's function-requests. The three buffer options available are discussed here.

1. Fixed Buffer Strategy - This strategy is used if neither of the two options below are indicated (i.e., this is the default strategy). Two buffers are allocated. For Read functions, one buffer can hold the Index ROOT block while the other is used for subsequent index levels and the data block itself. The two buffers are used effectively when the Block Split operation is required for file growth.

2. Buffer Pool - This strategy may be selected by the user. The user creates a Buffer Control Table (BCTBL using BCTGEN) which is filled in by OPEN and maintained by DMS. All indexed files per task may be accessed from the same buffer pool. Also, more than one buffer pool can be used within the same program.

   A buffer pool consists of a group of 3 to 60 buffers; each buffer is 2048 bytes (2K) and is acquired from the Segment 2/task using GETBUF. The BCTBL requires 56 bytes/buffer for buffer management.

There is no restriction of the order of opening and closing files using a buffer pool.

The Buffer Pool is managed using an LRU approach; all write operations are initiated directly (rather than being initiated at buffer replacement time, e.g., pageout by page). The internal features of DMS buffer pool maintainence are adjusted for performance in an interactive environment.

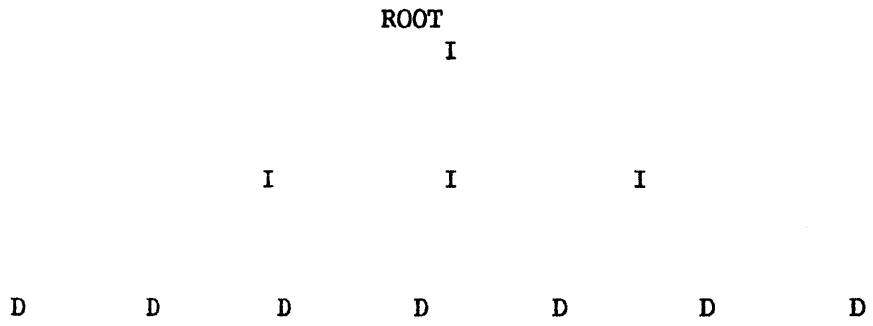
3. Large Buffer Strategy - This strategy may be selected by the user. In general, two buffers are assigned; however, one of these is a large buffer (large buffer size can be from 4K to 18K). This strategy is available for INPUT mode. For INPUT mode and sequential access, this strategy can save many IO operations.

These three strategies may be combined within a program if required. The use of buffer pooling for general indexed file random operations will certainly be helpful in significantly reducing the total number of I/O operations. The sharing task also makes use of the buffer pool to provide access to a number of SHARED files with efficient performance.

Indexed File Structure

Indexed files contain a number of 2K blocks. However, the relationship between blocks is established using chain fields and pointer fields. (In an indexed file, block (n) and block (n+1) are not necessarily accessed in order.)

Logical View of Blocks in a sample Indexed File



The above figure shows 3 levels. The first level is the ROOT block. The ROOT block is used when a primary key is supplied and a record is to be read. In the above figure, the ROOT contains POINTERS to 3 low-level index blocks. In turn, each of these index blocks contains POINTERS to the lowest level block - the structure. These lowest level blocks contain the data records in the indexed file.

The above figure shows 2 index levels and the lowest level of the structure (the record level). All blocks on one level are chained forward using a 3-byte CHAIN field at the end of each block.

Type I blocks (above) contain index items. An index item consists of a key field and a 3-byte pointer to a block on the next lower level. The key field in the item is equivalent to the highest key in the block pointed to.

Functional Overview Of Alternate Indexed File Support

Introduction

The alternate indexing facility for Wang 2200VS Indexed Files allows the creation of up to sixteen (extendable to 32 or 64) alternate access paths to records in a file. A program may specify on WRITE and REWRITE functions the access paths through which a particular record is to be accessible, and on READ and START functions may specify the access path to be used.

Each alternate access path is implemented as a tree-structured index defined on a field (starting position and length) within the data record, and referenced within the file's label (and by programs) as an ordinal between 1 and 16. The Data Management System maintains a map with each record of the file, indicating the access paths through which that record may currently be accessed.

New Fields for DMS Functions (UFB and AXD1)

In discussing the Data Management functions, fields have been added to the User File Block; also a new block (AXD1) is used. The AXD1 is described in detail later; for the purpose of this section, it can simply be regarded as a UFB extention-type block.

(1) AXD1MASK - A two-byte field into which a bit map is placed as a result of READ functions, indicating the currently available access paths for the record read. This field is to be set by the program before REWRITE or WRITE requests to indicate the access paths to be made available for accessing the record written. (A variable mask size from 1-8 bytes is a planned additional feature.)

(2) AXD1ALTINX - A one-byte field containing a binary number from 0 to 16. This indicates the access path (index structure) to be used on READ and START functions. Zero (0) indicates the primary index structure. One (1) through sixteen (16) select alternate index structures. The field contains 0 when the file is opened.

(3) UFBALTCNT - One byte, containing in binary the number of alternate index structures which can be processed using this User File Block. A file with more alternate access paths than is specified in this field will require that the Data Management System allocate additional storage (in the requestor's segment 2 buffer area) at OPEN time in which to keep alternate index structure descriptions. Compilers are expected to supply a proper value in this field and in UFBALTPTR, and to reserve sufficient space at the location addressed by UFBALTPTR to hold the index structure descriptions; they should not rely on the Data Management System to allocate buffer space, since that is a relatively inefficient use of memory. (Also see notes on AXD1-AREA at end of chapter.)

(4) UFBALTPTR - Three-byte address of an area (User File Block appendage or other segment 2 area) to contain alternate index structure descriptions (from AXD1ENTRY). If zero, space in the segment 2 buffer area will be allocated.

DMS Functions

READ Function

A READ by key value uses AXD1ALTINX to determine the proper access path. If the requested record can be found, it is placed in the record area. AXD1MASK is set to indicate the available access paths to that record. If duplicate key values are allowed on the specified access path, all other records with the same key value can be retrieved by repeatedly issuing READ NEXT requests. AXD1ALTINX is unchanged. If the access path specified by AXD1ALTINX does not exist, an invalid key condition occurs (as if the access path existed, but no records were accessible through it).

A READ NEXT uses AXD1ALTINX to determine the current access path. It reads the next record in sequence by the key field associated with that access path. (Immediately after OPEN, this is the first record on the primary access path.) AXD1MASK is set as for READ by key. The established key of reference access path corresponds to the AXD1ALTINX path (where AXD1ALTINX was set by a previous function request). If AXD1ALTINX is changed by the program between an OPEN, READ, or START, and a following READ-NEXT, that READ-NEXT will fail and 'Invalid Function Sequence' will be reported in the UFB file-status bytes; in this case, the UFB error exit is taken if present.

Invalid-Key and End-Of-File conditions occur under the same circumstances as for primary key access. (Notice, for example, that if 10 records are accessible through the first alternate access path of a file, End-Of-File status will be set after not more than 10 successive READ NEXT requests on that access path, no matter how many records are in the file.)

WRITE Function

A WRITE request does not use AXD1ALTINX, but sets it to zero (indicating the primary access path) when the WRITE completes. The new record becomes accessible through alternate access paths corresponding to the bits on (1) in AXD1MASK. If bits not corresponding to any defined access path are set, the WRITE operation fails with file status indicating 'Error, Invalid Mask' and the error exit is taken. Unless duplicate key values are allowed for an access path, an attempt to WRITE or REWRITE (IO mode) a record containing such a value results in the same file status as for a WRITE with duplicate primary key (22), and the entire operation fails. (The file is not changed.)

Duplicate key errors for output mode are not detected until CLOSE, at which time only the first of the records involved in the duplication is recorded in the alternate index structure for this access path and the CLOSE proceeds; the file status in the UFB is set to 'Invalid Key, Duplicate (22)' and is available there for examination by the program after CLOSE (an error-log file can be optionally created; this feature is described later).

REWRITE Function

REWRITE does not use AXD1ALTINX. It affects the current record, after READ WITH HOLD. The replaced record becomes accessible through the alternate access paths corresponding to the bits on (1) in AXD1MASK (as for WRITE). The record is no longer accessible through the access paths corresponding to zero (0) bits in AXD1MASK. (In this sense, REWRITE is equivalent to DELETE followed by WRITE.) The REWRITE operation fails with 'Invalid Mask' if any invalid mask bits are set (as in WRITE), or with 'Invalid Key, Duplicate (22)' if an alternate key field for which duplicates are not allowed contains a duplicate key.

DELETE Function

DELETE affects the current record, after READ WITH HOLD. AXD1ALTINX and AXD1MASK are not used. The deleted record is no longer available on any access path.

START Function

AXD1ALTINX determines the key to be compared for equal (eq), greater-than (gt), or greater-than-or-equal (ge) conditions. When duplicate keys are allowed on the specified access path, START positions the conceptual current-record-pointer to the first of these, so that successive READ NEXT requests will retrieve them all in primary key order.

OPEN Function - Existing File

AXD1ALTINX is zero after OPEN, thus indicating that the primary key is the current key of reference, and AXD1MASK indicates the existing index structures (as indicated in the file label). A READ NEXT retrieves the first record in primary key sequence.

OUTPUT Mode File Attribute Specification

UFBALTCNT and UFBALTPTR must be supplied in OUTPUT mode. UFBALTPTR must address an AXD1 control block with AXD1MSIZE supplied (2 in binary for the first implementation), and containing alternate index structure description entries (as defined in control block AXD1) for each alternate access path to be defined.

The key field ordinal (AXD1XORD), key location (AXD1KEYPOS), key length (AXD1KEYSIZE), and selectable options (AXD1EFLAGS) must be supplied in each entry. Selectable options in AXD1EFLAGS are: duplicate keys allowed and key compression in index structure.

Alternate File Errorlog (OUTPUT Mode)

When an alternate indexed file is created in OUTPUT mode, the user program presents records in ascending primary key sequence to DMS. (Each record has an associated user-supplied access mask.) The primary index structure is created as records are added. The alternate key information is held in a group of work records until the file is closed.

At SVC CLOSE, a program (BUILDALT) is called to create the alternate tree structures using these work records. The work records are sorted and the alternate tree structures are built.

Only during this last phase is it possible to detect the following error condition:

An alternate access path which does not support duplicates has duplicates (i.e., two or more records have the same alternate key value for the particular access path).

In this case, the record with the lowest primary key value will be accessible through the access path while other duplicates are not. (However, no masks in the records are updated.)

If this error occurs, the program (BUILDALT) will issue a message giving the user the choice of creating an error log file or not.

If an error-log file is to be created, BUILDALT will use filename = 'ERRORLOG' in the library (and volume) containing the alternate file. This consecutive file will contain fixed length records in the following format:

BYTES 1-2 = 2 unpacked decimal digits representing the ordinal number of the access path (e.g., X'3031').

BYTE 3 = ASCII blank (X'20').

BYTES 4-N = primary key value of record in error (i.e., record not accessible through the access path).

Using The Error Log To Correct Errors

1. Records in error can be rewritten with appropriate mask and/or alternate key values. This would require a separate user program (in COBOL).

2. If the access path definition was in error (i.e., duplicates should have been allowed), then the file should be re-created using a correct definition.

Notes:

1. Getting an error-log is always recommended.

2. The alternate file can be read sequentially with the access masks intact in order to recreate the file. All functions on the file will perform correctly although records in error will not be accessible through access paths where an invalid duplicate condition occurred.

3. Fixing all errors before further use of the file is recommended.

Overview Of Indexed And Alternate Indexed File Structures

Indexed Files

- Consist of a primary-tree structure with data blocks at the lowest level.

- Each data block contains records in ascending key sequence; the data records may be in one of the 3 DMS-record formats.

- Each data record contains a unique fixed-length imbedded primary key.

- Blocks on levels other than the lowest level are index blocks; these index blocks have item size = PK size + 3 (a 3 byte pointer to a block on the next lower level).

- All blocks in the primary tree (or any alternate tree) structure have a 2-byte block length (BL) prefix (DISP=0) and a 3-byte chain field (DISP=2045).

Alternate Indexed Files

- Contain a primary-tree structure.
- Contain one or more alternate tree structures.
- Contain a label block for alternate tree description and processing.

The primary-tree structure is the same as for indexed files except that each data record has a 2-byte access mask.

Internal Representation Of Low-Level Of Primary-Tree (Data Records Within Data Blocks)

The record size in the file label (FDR1RECSIZE) does not include the mask length. Internally, however, the mask is always stored with the record and is reflected in the RL prefix for variable-length records or an adjusted internal record size (=RECSIZE plus MASKSIZE) for fixed length record processing.

The access mask indicates through which alternate paths the record may be accessed. The access mask is not referenced as part of the record; it is present in AXD1MASK (through UFBALTPTR) and is never present in the user-record area. For variable-length data records, the record size supplied in UFBRECSIZE by the user (for REWRITE or WRITE) does not include the mask length; likewise, the value returned in UFBRECSIZE by DMS after a READ request does not include the mask length.

There is one tree structure for each alternate index (access path). The lowest level of an alternate tree structure contains items of the form:

Alternate Key Value (AK)	Primary Key Value (PK)

This item indicates that the data record with Primary Key = PK has the particular alternate field with value = AK. This fixed-length item will be used for the initial implementation (see note 2 below).

Blocks on levels other than the lowest level are index blocks within the alternate-tree structure. These index blocks have item size = AK size +3.

An alternate indexed file also has an alternate index descriptor block (AXD1) as the first block in the file (block number 0).

Initial Implementation Notes:

1. For access mask = 2 bytes, 16 (max) alternate trees are allowed. All DMS related control blocks contain space for possible expansion to access mask size of 4 or 8 bytes (with 32 or 64 maximum alternate trees).

   The access mask size is defined as 2 bytes for initial implementation.

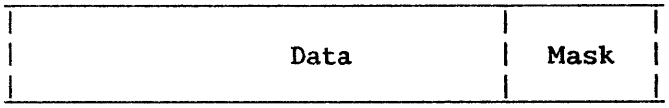
2. The low-level items are fixed length for a given alternate tree (size = PK size plus AK size); this fact is indicated in the information describing that tree. Variable-length, low-level items are a later extention (VLEN items will be useful for duplicates).

Internal DMS Record Formats For Alternate Indexed Files

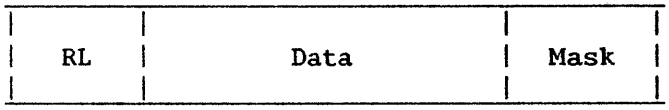
The internal DMS record formats for data records and for work records are briefly listed here. Naturally, the formats for data records apply to all data records in the file regardless of open mode.

The three DMS record formats are listed here for alternate indexed files. Note that the access mask is stored at the end of the record and that the mask is always maintained with its associated data (i.e., the mask length is included in the RL prefix for variable length records).

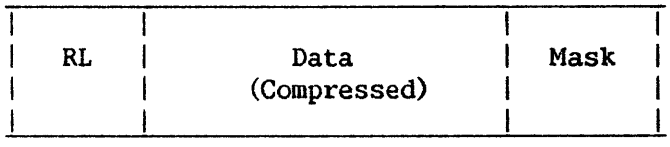
Fixed Length Format



Variable Length Format



Compressed Format



Note that for compressed format, the length of the compressed data is RL minus 2 minus mask-size; the compressed data begins at RL+2 as usual. Note that the mask is not compressed.

Work records are created from data records during OUTPUT mode DMS WRITE functions. These work records are sorted during CLOSE processing and used to create the low-level part of each alternate tree structure.

Work Record Format

XORD	Alt-key value	Primary Key Value

The length of each work record = 1 plus max (alt-key size) plus primary key size. (Stored in AXD1ORECSIZE.)

XORD refers to the ordinal tree structure number (AXD1ORD). For a work record containing an alternate key smaller than the maximum size, the work record contents beyond the end of the primary key field are undefined. Using fixed-length work records allows all records to be sorted on the same record size.

SVC OPEN - Existing Alternate Indexed File

The user supplies an AXD1-Area addressed by UFBALTPTR; the length is indicated by UFBALTCNT (length = (AXD1ENTRY-AXD1BEGIN) plus (UFBALTCNT * L'AXD1ENTRY)). If either field is zero (or if the area is invalid as flagged by MCBRWTST), then no user-supplied AXD1-Area is used by SVC OPEN. This case is not an error; SVC OPEN allocates the appropriate AXD1-Area from the user buffer space. (No AXD1-Area is used for BAM or PAM.)

The SVC OPEN requires almost no changes from normal indexed file support until the end of the routine when the UFB is set up. At this point, the appropriate DMS function vector addresses are placed in the UFB; also a third buffer is allocated (AXD1BCB). Block 0 in the file is the AXD1-Block; this block is read to a buffer area by SVC OPEN and then it is moved to the AXD1-Area. The information in each AXD1ENTRY ('label' information) is verified as in output mode (key size, key position and ordinal tree value). AXD1MASK is also verified as follows:

a. Create existing tree mask while checking ordinal number values (for duplicate or invalid ordinal tree structure-AXD1XORD).

b. This existing tree mask must equal AXD1PMASK.

If AXD1PMASK is invalid, or if any field within an AXD1ENTRY is invalid, a cancel message indicating invalid label information is displayed. The AXD1-Area which is loaded is either user-supplied or allocated directly by SVC OPEN.

After SVC OPEN, the following fields are set:

a. AXD1ALTINX = 0; AXD1CURINX = 0
*b. AXD1MASK = AXD1PMASK (indicating all trees) (AXD1PMASK is validated as part of label information verification)
*c. UFBALTCNT = FDR1ALTCNT (this is true after BAM and PAM OPEN also.)
d. UFBALTPTR = address of AXD1-Area
e. AXD1UFB = address of UFB
f. AXD1BCB = buffer address and OFB pointer set

For SHARED MODE, any user-supplied AXD1-Area is ignored.

*After an alternate indexed file is opened in SHARED MODE, UFBALTCNT is zero and AXD1MASK is zero in the user's AXD1-Area (from UFBALTPTR).

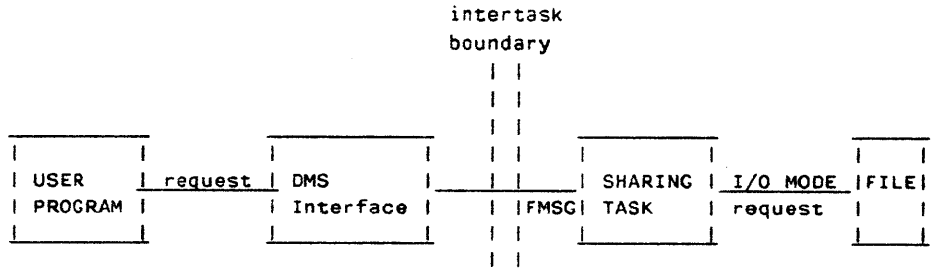
7.4 SHARED MODE

SHARED MODE is an OPEN MODE which allows multiple user programs (in effect) to access the same file through IO MODE type functions (i.e., multiple realtime requests for record addition, deletion or modification are supported.) The function-requests available to the user-program in SHARED MODE are identical to those in IO MODE (some situations do occur in SHARED MODE, based on the need for exclusion between concurrent requests for the same record, which do not occur in IO MODE). The user may have multiple files open in SHARED MODE.

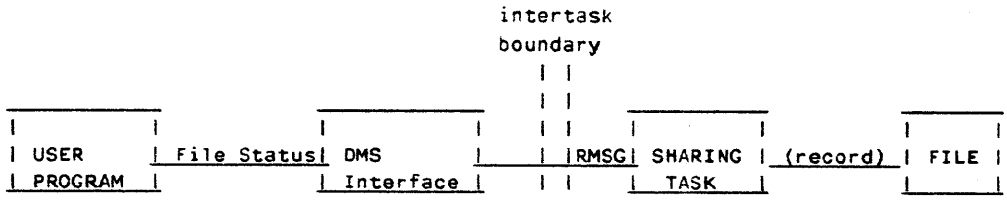
SHARED MODE is implemented on the 2200VS System through a SHARING TASK. This task has exclusive control of the file; it processes requests from user-programs one at a time and resolves any record conflicts. This task uses IO mode functions (with buffer pooling) for all file access. A mapping facility is provided by the 2200VS operating system (see SVC DTI) to allow efficient record transfer between a user task and the SHARING TASK.

The SHARING TASK is a separate task (which does not require a workstation). It has its own Segment 1 and Segment 2 and runs unpriviledged. It (currently) has no special privileges (i.e., it must adhere to all the restrictions of unprivileged operation). The only distinguishing feature it has is a port for intertask messages whose name is "@SHR". SVC OPEN and SVC CLOSE identify the SHARING TASK solely by PORT=@SHR.

The two following figures show the general steps in a request for one user accessing a record in one SHARED file.



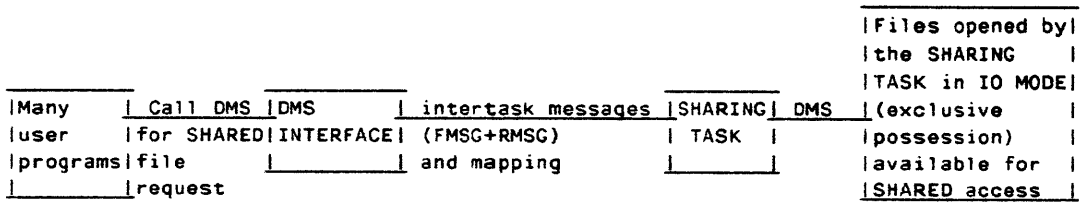
The user-program's request is packaged as a function message (FMSG) by the 'called' DMS interface vector. (Interface vector addresses are placed in the user's UFB by SVC OPEN.) The FMSG is sent to the SHARING TASK. The SHARING TASK issues corresponding IO MODE request(s) using DMS on the file.



The SHARING TASK receives file status as a result of the IO MODE operation. The SHARING TASK sends a response message (RMSG) to the interface vector. The interface vector moves the file status to the UFB. All OPEN, DMS, and CLOSE messages use the same message format: function message (FMSG) and response message (RMSG).

The steps for MAPPING THE RECORD AREA involve virtual memory mapping such that the record area in the SHARING TASK Segment 2 (which, for example, receives a record on a READ request) can be mapped to (and from) the USER'S SEGMENT 2 without any actual data movement.

In a general environment, we have the following:



The number of user programs able to have SHARED FILES open, the number of SHARED FILES per user, and the total number of SHARED FILES supported by the SHARING TASK are restricted only by the amount of space available in the SHARING TASK's segment 2 for control blocks and mapping areas.

In this structure, a high degree of protection is automatically achieved since:

1. User programs cannot access the file directly.

2. File buffers are in the SHARER'S segment 2 and are unavailable to any user program.

The HOLD mechanism in SHARED MODE - The READ HOLD function-request is required if a record is to be rewritten or deleted. The HOLD mechanism in SHARED MODE is maintained by the SHARING TASK. The SHARING TASK allows each user program to HOLD one record (in any shared file) through READ HOLD. While this record is HELD, no other user may access it with READ HOLD; i.e., other READ HOLD requests will be forced to WAIT until the record desired is released by the user.

A HELD record in a file is released (removed from HELD state) only when one of the following occurs:

1. The user (holding the record) issues another READ HOLD on any SHARED file

2. The user issues a REWRITE, DELETE, or START RELEASE on this file.

3. The user CLOSES the file or abnormally terminates processing.

The HOLD mechanism provides lockout on a record basis so that concurrent requests for the same record can be handled correctly. A program should attempt to HOLD records for the shortest possible time in order to improve concurrency with other users. Extensions to provide a fuller capability for multiple record updates are to be added at the SHARING TASK level.

Additional functions available to the user program in SHARED mode are:

1. START RELEASE - This function is available to specifically release a HELD record (or HELD file as below). Thus, the general rule for avoiding record conflict is that a READ HOLD should be followed by a REWRITE, DELETE or START RELEASE as quickly as possible.

2. START HOLD - This function allows a user to gain control of a file for the purpose of modifying several records or insuring that the file remains consistent while an update is made. It insures that no records are HELD by other users. This function naturally increases the possibility of forcing other users to wait. The FILE HOLD is removed by START RELEASE. A START HOLD is invalid if issued when already holding any record or any file.

The above functions are considered as null operations if issued in IO mode (i.e., at the DMS level).

The READ KEYED and START KEYED functions in SHARED Mode return with FS = 80 if UFBKEYAREA does not point to the key embedded in the record.

Log Files

A Log-File is a consecutive file which can be opened in SHARED mode. A Log-File has records in variable length or compressed format.

When a consecutive file is opened in SHARED mode, the sharer will use an existing Log-File (in EXTEND mode, if found). Otherwise, the sharer will create a Log-File with the user-supplied file name. When a Log-File is created, the compress flag and the record count can be supplied by the user program (default record count = 1000). The record size for the file (maximum) must be supplied when the file is created. (When a Log-File is created by the sharer, file protection class is set to blank (any user); Log-Files created by a user program and subsequently used in SHARED mode will have normal file protection class validation.)

Multiple users can write records to a Log-File once it is opened in SHARED mode. Each record is added at the end of the file. The sharer will support more than one Log-File opened at a time. Log-Files are accessed as normal consecutive files (with variable-length or compressed records) in INPUT, IO, OUTPUT, and EXTEND modes.

Log-File Special Features:

1. Recovery Feature - A Log-File will be opened successfully even if it was not closed due to a system crash. The recovery is automatic during the next SVC OPEN issued on the file (in any mode). Both record count and EOF indicator are recovered; no message is issued.

2. Write-Through Feature - Even with recovery, the last block of data in the DMS buffer will be lost if the system crashes. This can be avoided if a disk write operation is performed for every logical record WRITE operation (i.e., the Write-Through feature).

If the first character of the filename = C'@', then the Write-Through feature will be in effect for all writes to that file (for OUTPUT, EXTEND, or SHARED mode).

Read-Only Access In SHARED Mode

Opening a file for shared update requires full (write) access rights. This new feature will allow a user having only read-access rights (to a file) to open the file in SHARED mode. However, the sharer will flag as invalid any REWRITE, DELETE, or WRITE function by that user. (All READ and START functions are allowed.) A file status value of 85 is returned if such an invalid function is attempted.

Advanced Sharing (Multiple Resources)

The current sharing facilities (e.g., READ-HOLD, START HOLD-FILE, START RELEASE) will be expanded to support the following additional features:

1. Two hold classes (hold for update/hold for retrieval)
2. Explicit resource control by range or list
3. Extension rights
4. Timeout option for wait on resource

These new features will be available using the standard DMS START vector and a UFB timeout field (value/exit-address) for SHARED mode processing. The features are designed for ease of use by the application programmer while providing powerful functions and being expandable. A general description of each of the features is provided below. These new features apply to both indexed files and alternate indexed files opened in SHARED mode. These features do not apply at all to log files opened in SHARED mode.

1. Hold For Update / Hold For Retrieval

When Hold For Update is used, no other user will be allowed to hold the particular resources for either update or retrieval until the current holder releases them. Hold For Update is always implied when a READ-HOLD function request is issued; in this way, the user can then rewrite the record without conflict in the shared environment.

When Hold For Retrieval is used, more than one user is allowed to hold the particular resources; however, a file or record held for retrieval will not be granted to a user who wishes to hold it for update. Hold for Retrieval is useful when a file is to be scanned (for a report for example) and no updates are to be made on the file during the time of the operation.

Switching between the two hold classes is not supported under advanced sharing. When a user is 'explicitly' holding a resource for update, a request to hold that same resource for retrieval is always flagged as an invalid function sequence. The same is true when a user 'explicitly' holding a resource for retrieval requests that it be held for update.

2. Explicit Resource Control By Range Or List

A resource may be held 'explicitly' without extension rights by using one of the following function requests:

START HOLD FILE - Hold the whole file
START RANGE - Hold a range of records (primary key)

These functions support hold for update or retrieval (set by user; default = update). They also support a list option such that repeated requests to hold file or range are gathered into a list by the sharer; the resources are only obtained (held) when the user issues a request to end the list and hold all the resources in one operation. Thus the list option allows any variety of resources within multiple files to be held in one operation; since the operation is indivisible with respect to other hold requests, deadlock cannot occur.

The START RANGE request and the list option are discussed in detail in the following paragraphs.

a. The START RANGE Request - this request does not affect file position. The request describes a range of primary key values by supplying parameters in the UFBKEYAREA and UFBGKSIZE. UFBGKSIZE is used to specify the effectve key size of the range; a value of zero or a value greater than primary key size causes the primary key size to be the effective key size for the range.

If the effective key size is the primary key size, then the range contains only one record (since primary keys are unique). The following examples illustrate the range specification rules: (GK = UGBGKSIZE, PK = primary key size, let key field contain 'ABCDEFG').

1. PK=7, GK=4 Range = all PK where first 4 chars = ABCD
2. PK=7, GK=2 Range = all PK where first 2 chars = AB

When a range is specified, no record may be added within that range by a WRITE request from another user. Thus the range notation is similar to the hold file concept where a range is merely a contiguous logical subset (by primary key value) of the whole file. When ranges overlap for the same hold class, the largest limits of the range are used. When ranges overlap with opposite hold classes, then the last function (which caused the overlap) is invalid.

It is possible and even practical to hold a range which contains no records. The number of records within a range does not affect the function of the range request or other DMS requests.

b. The List Option - The list modifier bit is available with the START HOLD-FILE and START RANGE function requests. A value of zero indicates the end of the list (default value). A value of 1 indicates that the current hold request should be added to any pending hold list for this user. The pending hold list will expand until a START function with End-Of-List causes all resources in the list to be held. As each entry is added to the list, it is checked against the other list entries for validity (no overlap of different hold class on the same resource) and for compaction (if several entries on the list may be combined). The order of the list is never of any significance since all resources on the list must be acquired together.

When a list is pending, it will remain pending until it is ended by a START HOLD-FILE or START RANGE with list = 0 (END) or until a START-RELEASE (or SVC CLOSE) is issued on any shared file by this user. (START RELEASE (ALL) and SVC CLOSE cause the pending list to be erased).

All DMS functions other than START (as above) are flagged as invalid if issued while a list is pending.

3. Extension Rights

Extension rights are requested using the START-EXTENSION-RIGHTS function request. In order to avoid the possibility of deadlock, only one user at a time may have extension rights and a user may not hold any resources when the request for extension rights is issued.

START XRIGHTS may be issued while 'implicitly' holding a record; in this case, the function is valid and the record is automatically released at the beginning of the function request.

Extension rights provide the only way a user may acquire more resources when already holding resources. When a user has extension rights, he may issue READ-HOLD, START HOLD (file) and START RANGE in order to accumulate as many resources as desired. Each resource request is performed immediately when issued while having extension rights (the list option is ignored when extension rights are active).

Immediately after the user has obtained all the desired resources, extension rights (only) should be released so that another user might acquire these rights.

4. Timeout Option

A user-supplied error exit specifically for timeout while waiting for a resource will be provided. A specific file status value for timeout will be used and R0 will contain the address of the function request if the timeout exit is taken (same as EODAD/ERRAD exit).

Note - File position remains unchanged after a time-out exit has been taken (i.e., a READ-NEXT HOLD with time-out could be repeated if the record was unavailable when first issued).

The UFB fields UFBTIME (1 byte for timeout wait period in seconds) and UFBTIMEEXIT (user supplied exit address) will be available in order to set a time value. Time values will remain in effect for all function requests until changed by the user-program. A time value = 0 implies an immediate return through the timeout exit if the resource is unavailable (i.e., wait zero seconds). If UFBTIMEEXIT (timeout exit address) is zero, then no timeout option is to be used (i.e., the program will wait as long as required until the desired resource becomes available).

The timeout exit may be taken as a result of any of the following DMS functions issued on a file opened in SHARED mode:

1. READ HOLD (next or by key)
2. START HOLD-FILE, START RANGE
3. START XRIGHTS
4. WRITE

(The timeout exit would be taken whenever the resource required was unavailable within the given time period (UFBTIME) and UFBTIMEEXIT was not equal to zero).

The timeout option should be very useful and easy to use for application programmers. Other fields in the UFB have been provided so that the resource held and the current owner are available after a timeout exit has been taken. This information allows the application programmer to create an exact message to the user whenever a resource is unavailable. For example, a message containing primary key value and current owner's initials might be:

"THE JONES RECORD IS BEING USED BY ABB"

New UFB fields (SHARED mode) for time-out option:
(Note - these 3 fields are all set = 0 by SVC OPEN.)

1. UFBTIME - One-byte wait time in seconds (0 = no limit)
2. UFBTIMEEXIT - Exit address for timeout return
3. UFBHOLDID - Initials of holder of resource

When a timeout exit is taken, a file status value of C'70' is set. UFBHOLDID is always set. UFBKEYAREA will be set with the primary key value of the record in conflict if the function was 'READ NEXT HOLD'. For all other functions, UFBKEYAREA is unchanged when the timeout exit is taken.

General Notes

1. The File-Hold Feature

A file may be held for retrieval or update by using the START HOLD function. Holding a file is a logical concept and is not equivalent to holding all the records in the file individually. When a file is held for retrieval, a WRITE function on the file will not be executed until the file is released. When a file is held for update, only the user holding the file may issue a WRITE (to add a record to the file). When holding a file, attempting to hold a range of records within the file (in the same hold class)* has no affect. Likewise, a READ HOLD issued on a file one holds for update will proceed without affecting the hold status on the whole file. (A subsequent REWRITE or DELETE would also not affect the hold status on the whole file.)

* If the hold class for the range is different than that for the file, the start range function is flagged as an invalid function-request. Also, a START HOLD-FILE request would be flagged as invalid if issued while holding a range within that file of the opposite hold class.

2. Releasing Resources

When a record is held for update by issuing a READ-HOLD function request, the record is said to be held 'implicitly'. In all other cases, a resource record (or file) is said to be held 'explicitly'. (If a user holds a file or range and issues a valid READ-HOLD, or if a user issues a READ-HOLD function while having extension rights, the resource is still considered to be held 'explicitly'.)

If a record is held implicitly, it will be released by any one of the following actions:

a. Issue REWRITE or DELETE on held record
b. Issue another READ W/HOLD on a shared file
c. Issue any START function other than positioning
d. Issue CLOSE (this file)
e. Issue WRITE on any shared file

Case B above must include the automatic release of the 'implicitly' held record before the other record is read with hold. For cases A,C,D,E, no resource is held after the request completes. By definition, if a record is held 'implicitly', it will be the only resource held by the program at that time.

If a resource is held 'explicitly', it must be explicitly released by using START-RELEASE or closing the file. The START RELEASE-FILE function is also available; it releases all resources within one file only. (Closing a file also causes release of the resources in that file only.)

If a program fails to release an explicitly held resource before issuing a READ-HOLD, etc. for another resource, this would be flagged as an invalid function (assuming the program did not have extension rights). This means that there is never any automatic release for an explicitly held resource.

Detailed Functional Overview - SHARED Mode

The following function descriptions assume that the user program does not have extension rights.

1. READ HOLD - A READ HOLD involves holding one record for update. A READ HOLD may have to wait if the required record is held by another user.

A READ HOLD may be issued when holding no resources. In this case, the record will be held for update by the user after the function-request completes. If a READ HOLD is issued while 'implicitly' holding any record, that record will be automatically released at the start of the function request.

If a READ HOLD is issued while 'explicitly' holding one or more resources, several situations may occur. If the user is not holding the indicated record for update, then the READ-HOLD is an invalid function (the user may be holding the whole file (or an appropriate range) for update in which case the READ-HOLD is valid). If the record is held for retrieval by this user, the READ-HOLD is flagged as an invalid function.

2. REWRITE and DELETE - These functions require that the record to be processed be held for update. The record will be automatically releasd after the function if the record was held 'implicitly'; the record will remain held for update if it was held 'explicitly'. There is never a wait for resource for these functions.

3. READ NO-HOLD and START For Positioning - These functions are not affected by the sharing mechanism.

4. WRITE Function - A WRITE function may be successfully issued when holding no resources. The WRITE function may have to wait if the file is held (for retrieval or update) by another user; a wait is also required if the new record would fall within the range of records held by another user.

If a WRITE function is issued while 'implicitly' holding a record, that record will be automatically released at the start of the WRITE function-request. A WRITE function-request will be flagged as invalid if it is issued while 'explicitly' holding any resources (unless the file involved or an appropriate range (held for update) is included in those resources).

5. Resource Control Commands (START Function) - All resource control START commands are valid if issued while holding no resources; the user program must wait if the resource request conflicts with resources held by other users.

All resource control START commands cause an automatic release of any 'implicitly' held record.

The START HOLD commands (with or without the list option) are invalid if issued while 'explicitly' holding any resource.

The START RELEASE (ALL) and the START RELEASE-FILE commands are always valid; no operation occurs if no resource can be released.

The following functional descriptions assume the user has extension-rights. Under extension rights, the user can continually hold more resources; all resources are released along with extension rights when the program issues a START RELEASE or terminates.

1. READ HOLD - The READ HOLD function can be used to hold records explicitly (for update) while the user has extension rights. If the user already holds the record for update (e.g., by holding the file or a range containing the record), the record remains held within the larger group and the READ HOLD is successful. If the user holds the record for retrieval, the READ HOLD request is flagged as an invalid function. The user may have to wait if the record is held by another user.

2. REWRITE and DELETE Functions - These functions must be issued on a record which is held for update. When the user has extension rights, there is no 'automatic release' on the record after a REWRITE or DELETE (i.e., the record was held explicitly).

3. WRITE Function - The WRITE function may be issued at any time if the user has extension rights; the only error condition results if the file (or record range) is held for retrieval by the user.

4. Resource Control Commands (START Function) - These commands can be issued while having extension rights. Overlapping requests are handled such that more resources may be obtained; requests for records already held have no effect if the hold class agrees. Any request which would require changing a hold class for resources already held by this user is flagged as invalid (i.e., changing between update and retrieval classes is not supported). The START EXTENSION-RIGHTS command is also flagged as invalid if issued when the user already has those rights.

Summary Of START Functions ('#' Indicates New Feature)

START EQUAL, START EQUAL OR GREATER, and START GREATER THAN positioning functions are unchanged.

START HOLD (80)		Hold file for update
START HOLD RETRIEVAL (CO)	#	Hold file for retrieval
START RANGE (84)	#	Hold range for update
START RANGE RETRIEVAL (C4)	#	Hold range for retrieval

(The 4 functions above have the list option also (10).)

START RELEASE (20)		Release all resources (this user)
START RELEASE FILE (24)	#	Release all resources in this file
START RELEASE XRIGHTS (21)	#	Release Xrights only
START HOLD XRIGHTS (81)	#	Obtain extensions rights

UFB Field Updates

The new fields UFBTIME, UFBTIMEEXIT, and UFBHOLDID will be added for SHARED mode only.

The following equated values will also be added:

START modifier bits indexed files -

Existing values remaining unchanged are UFBVEQ (01), UFBVGT (02) and UFBVGE (03). The following two values are unchanged although their meaning is slightly altered.

UFBVHFILE	(80)	START function = HOLD
UFBVRLS	(20)	START function = RELEASE

New Values:

UFBVRANGE	(04)	Request is for a range
UFBVRETRIEVAL	(40)	Hold class = retrieval if set
UFBVXRTS	(01)	Extension rights for hold/RLS
UFBVLIST	(10)	List option (hold range or file)

New File Status Values:

UFBFS1TIME	EQU	C'7'	FS value for time-out on resource
UFBFS1SHARE	EQU	C'8'	FS class for sharer conditions
UFBFS2ACC	EQU	C'5'	Update access denied (FS=C'85') for user with read-only rights in SHARED mode
UFBFS2RESERR	EQU	C'6'	Resource control error (FS=C'86')

In using FS=86 for all invalid resource control commands, some FS=95 should be updated. FS=95 for record not held would remain valid for IO mode; however, for SHARED mode this case would be changed so that FS = 86 was used.

7.5 DMS FUNCTION-REQUESTS

This section contains a brief list of the UFB values used by DMS. An extensive list of all DMS filestatus conditions is also included, along with a description of the various DMS error returns.

7.5.1 DMS Function-Request Entry

The performance of a DMS function-request on a file is effected by calling a DMS (function-request) routine. The DMS routine addresses are available in the UFB (after OPEN). The DMS routines define the access method to be used in processing the file; functions not supported for a particular filetype OPEN mode combination are indicated by a DMS routine address (in the UFB) which returns an INVALID FUNCTION error indication.

DMS routines are entered using the CALL macroinstruction (JSCI) with the following registers loaded:

a. R1 - UFB pointer
b. R15 - Stack pointer (updated by CALL)

Several standard operations are taken at the start of all DMS routines.

a. Establish addressability, etc.

b. Verify that the file has been opened by checking that OFBUFB (UFB pointer from indicated OFB) equals UFB pointer in R1. (This check insures that a valid OFB exists. Failure will cause a 'fatal error' with a meaningful error indication.) The OPEN Mode (processing mode) is established through OFBFLAGS.

The following UFB fields are generally used during a function-request. The specific UFB fields per function-request are described later.

i. UFBOFB - Pointer to OFB for this UFB.

ii. Modifier byte from function vector - used by some DMS routines.

iii. UFBRECSIZE - Length in bytes used for all record movement and for maintaining the sequential record pointer (SRP).

iv. UFBBUFDATAL - Length in bytes for physical I/O data transfer.

v. UFBRECAREA - Pointer to user-record-area (loaded from UFB for each function-request).

vi. UFBBCBFLAGS, UFBBUFOFFSET - buffer control information.

vii. UFBEODAD, UFBERRAD - Error and exceptional condition exit address (loaded from UFB as required).

viii. UFBFS1, UFBFS2 - File Status (bytes) set by all DMS routines on return.

ix. UFBLF - Last function on file, loaded by each DMS routine (also UFBLFMOD - last modifier byte).

x. UFBKEYAREA - Pointer to user supplied key or relative record number.

Common UFB Input Parameters - Available for all consecutive disk file operations.

UFBEODAD	(May be supplied by user prior
UFBERRAD	to function-request)
UFBLF	
UFBLFMOD	
UFBBUFSIZE	

The following UFB fields are used as 16-bit positive numbers rather than halfwords (15 bits and sign). These fields are used only by DMS for most cases: UFBBLKSIZE, UFBBUFSIZE, UFBCHKSIZE, UFBBUFDATAL, UFBBUFSIZE, and UFBMAXTFR.

7.5.2 DMS Function-Request Return

DMS returns to the user program using the RETURN macro-instruction. User registers 2 through 15 are always restored. Register 0 (R0) is restored unless UFBEODAD or UFBERRAD is used -- R0 then contains the normal return address. Register 1 (R1) is restored unless the Read-No-Data option has been used, in which case R1 contains the record address.

DMS indicates the result of the function-request through the file status bytes UFBFS1 and UFBFS2. These bytes generally contain a value of X'30' - X'39' corresponding to the ASCII characters 0 through 9. This value is called the File Status (FS) Code. Refer to Appendix C for a list of the DMS and ADMS File Status Codes.

Return Address Usage (Return to User-Program)

1. Normal return - The RETURN instruction causes registers to be restored and the program continues at the instruction after the DMS call.

2. UFBEODAD return (UFBEODAD not = 0) - The address in UFBEODAD is used. The program continues at this address; file status is set to the appropriate value and R0 = the normal return address.*

3. UFBEODAD return (UFBEODAD = 0 and UFBERRAD not - 0) In this case, the address in UFBERRAD is used. The program continues at this address; file status is set and R0 = the normal return address.*

4. UFBERRAD return (UFBERRAD not = 0) - The address in UFBERRAD is used. Before returning to the program, an acknowledge-type GETPARM message is issued by DMS. This message indicates the UFB address, the DMS function-request address, the file status value and a brief description of the significance of the file status. The user may continue or CANCEL at this point. If the program is continued, return is made to the address in UFBERRAD with file status set and R0 = the normal return address.* If UFBF4NOACK is set, the acknowledge message is not issued and return is made using UFBERRAD.

* - The high-order byte of R0 is set to zero

DMS Fatal Errors

A fatal error causes DMS to issue an SVC CANCEL. The CANCEL message describes the situation and gives the UFB address, function-request address and PRNAME of the file. The user-program cannot be continued after a CANCEL is issued.

1. Fatal errors for file status error conditions - If UFBERRAD is zero (for file-status greater than or equal to C'30') or if both UFBEODAD and UFBERRAD are zero (for file-status less than C'30'), then the user has not supplied a return address for file-status error conditions. In this case, the CANCEL message will include the file status value and a description of the file status significance.

2. Other fatal errors - In the course of performing a function-request, DMS uses several UFB fields. If any of these fields is inconsistent or invalid, DMS will recognize this and CANCEL the program. The CANCEL message will indicate which field had the invalid value. These errors are generally caused by the user-program incorrectly modifying DMS fields in the UFB.

3. Program check during DMS function-request - If a program check occurs in DMS, the most likely reason is an invalid address in UFBRECAREA or UFBKEYAREA. (These fields are not checked since the general DMS requirement is efficiency.) A program check in DMS can be identified by inspecting the save area trace (available as a debugging command in HELP); register R1 in the DMS save area contains the UFB address.

7.6 PRINTER SUPPORT

Records output to a printer file are variable-length (P-type) records as described in Section 4.2.1. Data may be written to a printer using either the Record Access Method (RAM) or the Phsyical Access Method (PAM). The Write function-request under RAM is described in this section. The Physical Access Method (as described in Section 4.8) is available to write a block of P-type records to a printer.

The user program views a printer file as an output-only file containing an unlimited number of variable-length records. A printer file record contains a 2-byte user-supplied control field and a variable number of data characters (depending on record size). DMS routines are available to write a user record to the printer file. The maximum record size for the file is established by the value of UFBRECSITE at SVC OPEN. File Status = '97' (invalid length) is returned if the value of UFBRECSITE for a function-request exceeds this value.

A printer file always has UFBFORGPRINT set. (This implies that the user program has supplied the 2-byte printer control field as part of the record*.) UFBBUFDATAL (length used for physical I/O) is set from UFBRECSIZE. (UFBBUFDATAL is maximum 134 for a printer file using RAM.) For expanded characters, the device will truncate any line after 68 characters; no error reported.

The only valid OPEN mode for a printer file is OUTPUT mode. The only valid function-request for a printer file is WRITE.

* If UFBFORGPRINT is set for a disk file, a consecutive file with variable-length compressed records is created. A large buffer size may be set for a printer file prior to SVC OPEN. This buffer size will have no effect if the output is directed to a printer. However, if the sequential output is directed to a disk file, the large buffer will result in fewer physical I/O operations.

7.6.1 Write Function-Request (OUTPUT)

Inputs:
a. Common UFB Input Parameters
b. UFBRECAREA - Location of the user record area (any Segment 2 address).
c. UFBRECSIZE - Length of record. This length is moved to UFBBUFDATAL for each Write function-request.

DMS routine operations: The record (as indicated by UFBRECAREA and UFBBUFDATAL) is moved to a protected buffer (by SVC XIO) and then the printer I/O write operation is initiated. The operation is waited on (at the DMS level) before the next record is moved to the buffer. (The modifier byte is ignored.) Trailing blanks are stripped from the record by DMS before calling SVC XIO.

Since the file has been successfully opened there are no (I/O) error conditions returned to the user other than invalid length or invalid command (FS='96' or '97'). (Out-of-paper, deselected, power-off, or device not operational are handled by SVC XIO.)

Outputs: UFBFS1, UFBFS2 stored in UFB.
Record written to printer file.
UFBLF is set to indicate WRITE.

7.7 WORKSTATION SUPPORT

A workstation on the 2200VS system is defined as a CRT and a keyboard. All workstation processing by DMS is performed under the Record Access Method (RAM) in IO mode.

The user program views a workstation file as an interactive file containing a fixed number of records (i.e., CRT rows) of a fixed maximum length (i.e., CRT columns). A workstation record (in the user-record-area) contains a 4-byte order-area followed by a variable number of data characters (mapping area). Workstation records are written to the CRT by the user program. The user program can read CRT records which can be modified by the workstation operator (through the keyboard).

UFBBUFDATAL (length used for physical IO) is set from UFBRECSIZE. Workstation Read or Rewrite operations are performed directly on the record area in Segment 2. The record area must be word aligned; otherwise, File Status indicating the error will be returned. Each workstation has an associated unit control block (UCB) which contains the 'current' AID character and keyboard status (locked or unlocked).

A workstation file can only be OPENED in IO Mode. The function-requests available are READ, REWRITE, and START.

An order check is returned as C'34'. An order check is detected by examining the IOSW; however, it is not considered an I/O error for logging purposes. (Power-off, deselected or device not operational are handled by SVC XIO.) Other error conditions are incorrect length and record area not aligned (File Status = '97' and '96' respectively).

The input for workstation function-requests is the common UFB (workstation) Input Parameters. This includes the Common UFB Input Parameters, UFBRECAREA (an address pointing to the user-record-area), and UFBKEYAREA for address of word containing the Relative Record number (i.e., row number, not used for START function-request).

Workstation operations use UFBRECSIZE as the length of the record. UFBRECSIZE less than 4 results in the Order Check error condition. The user may set UFBRECSIZE before any function-request. UFBRECSIZE is moved to UFBBUFDATAL by the DMS vector routine before initiating the physical I/O operation.

7.7.1 Read Function-Request (I/O)

Inputs:
a. Common UFB (WS) Input Parameters
b. Modifier byte at UFBVREAD
   MODIFIABLE option
   TABS option (nonCOBOL)
c. UFBRECSIZE - Length of record

DMS routine operations: The relative record number (at the word pointed to by UFBKEYAREA) is moved to the first byte of the order area. (RRN other than 124 causes an order check return to the issuer.) If UFBKEYAREA=0, the row number already in the order area is left unchanged. The DMS routine then calls SVC XIO to initiate the read I/O operation (the modifier byte is reflected in the command byte sent to SVC XIO). SVC XIO will issue the command when the keyboard is locked (or when it becomes locked through operator action). After the wait for I/O completion, the following information is available:

a. The record (at order area +4) is available in the user-record-area.

b. Order area bytes 2-3 in the user-record-area receive the current cursor position.

c. UFBFS2 receives the AID character (from the UCB).

MODIFIABLE option* - This option causes the following actions:

a. The modifiable fields within the record are loaded from the corresponding workstation CRT positions.

b. Protected fields within the record are either skipped (leaving those relative locations within the user-record-area unchanged) or moved from the workstation to the user-record-area (depending on workstation model). In the later case, the protected fields will receive the characters present when that line was last written to the screen.

c. Pseudo-blank characters on the screen are changed to blanks before transmission; therefore, they are also represented as blanks within the user-record-area.

d. Blinking characters within the range of the READ operation will be changed to high-intensity nonblinking characters (by changing the associated field attribute character both on the screen and in the user's record area).

e. Order area bytes 2-3 in the user-record-area receive the current cursor position.

f. UFBFS2 receives the AID character (from the UCB).

Outputs: UFBFS1 stored.
UFBFS2 stored (equal AID character).
Order check (UFBFS1, UFBFS2 = 34) is a possible exception (return through UFBERRAD, no logging of 'I/O error').

Record is available in user-record-area.
UFBLF, UFBLFMOD set.

(Keyboard is locked.)

* The recommended sequence of workstation function-requests involves Read Modifiable rather than Read.

Recommended sequence (one user-record-area)

1. Rewrite record to workstation.

2. Read modifiable; data entered by workstation operator is now available in the user-record-area.

3. The user-program may change fields within the user-record-area based on the data received.

4. Rewrite record to workstation (for additional information, error correction (reentry), etc.).

It is important to note that protected fields within the user-record-area should be changed after the Read Modifiable function-request (rather than before the Read Modifiable function-request).

7.7.2 Rewrite Function-Request (I/O)

Inputs: a. Common UFB (WS) Input Parameters
b. Modifier byte at UFBVREWRITE
TABS option (nonCOBOL)
c. UFBRECSIZE - Length of record

DMS routine operations: The relative record number (at UFBKEYAREA, indicating row number) is moved to the first byte of the order area (user-record-area). (RRN greater than 24 causes an order check return to user.) If UFBKEYAREA=0, the row number already in the order area is left unchanged. The second byte of the order area is a user-supplied Write Control Character (WCC); the third byte is a user-supplied column number (for optional cursor positioning); these bytes are not used by the DMS routine.

The DMS routine initiates the physical I/O write operation and waits for completion. After completion, the DMS routine returns control to the user-program.

Outputs: UFBFS1, UFBFS2 set for operation successful. (Order check is also possible.)

The workstation record is written (including actions directed by the WCC). UFBLF, UFBLFMOD are set.

7.7.3 Start Function-Request (I/O)

Inputs: Common UFB (WS) Input Parameters

DMS routine operations: The DMS routine inspects the UCB for the current state of the workstation; it then sets the file status bytes accordingly.

UFBFS1 = C'0' - Operation successful
UFBFS2 = AID character - If AID character = blank, then the keyboard is unlocked; otherwise, the keyboard is locked and UFBFS2 = current AID character.

(The Start function-request can be issued (prior to a Read function-request) to determine whether a subsequent Read function-request would wait (for keyboard to be locked) or would process immediately.)

Outputs: UFBFS1, UFBFS2 set
UFBLF, UFBLFMOD set.

7.8 MAGNETIC TAPE SUPPORT

Wang magnetic tapes are processed through the Data Management System, using OPEN, DMS functions, and CLOSE.

Two tape densities, 800 and 1600 BPI, are supported. At tape mount, a physical I/O will test the tape density and indicate it in the UCB Tape Density Field. At OPEN, the density will be checked against the user-specified value.

Tape labels supported in the first release are ANSI-Label (AL), IBM-Label (IL), and No-Label (NL). They are described in our Tape Label Control Blocks. Tape labels are written in ASCII code for AL, and EBCDIC for IL. Tape data structure and labels are compatible with the industry standard (see the IBM OS Tape Manual, for example), with the following restriction and extensions:

(1) The label-handling routines will always skip existing User Header and Trailer labels, and will not create them on new tape files.

(2) Only 80-byte labels are accepted.

(3) Five kinds of record formats are supported:

'F' - fixed-length format
'V' - variable-length IBM format
'W' - variable-length Wang format
'X' - compressed Wang format
'U' - undefined-length format

Tape data blocks of format 'W' and 'X' are to contain a block length prefix (like the corresponding disk blocks) in order that short blocks (less than 18 bytes) can be padded to prevent their being interpreted as 'noise' blocks.

Tape files in 'V' format have IBM variable-length record blocking format. Short blocks are padded, as above, if less than 18 bytes long.

For 'U' format files, each block is considered as a record. Short 'U' format blocks are never padded.

Receipt of a block of less than 12 bytes (in INPUT mode) is treated as a length error.

(4) Minimum block size is 12 bytes for INPUT, and 18 bytes for OUTPUT.

(5) For IBM-label tape, EXTEND mode is not supported.

Tape files may be positioned by file-sequence number. The program specifies file sequence in UFBTSEQ before opening the file. For NL tape, each file is separated by tape marks. For an existing labelled tape file being opened in INPUT or EXTEND mode, positioning can also be done by library and file names (see below).

The Wang Data Management System supports tape access in INPUT, OUTPUT, and EXTEND modes, through the RAM, BAM, and PAM access methods. CLOSE allows a user to rewind, rewind and unload, or not rewind the tape.

For an existing labelled tape file, DMS will use label information to determine the buffer size. For a new tape file or NL tape file, the program must supply the buffer size in UFBBLKSIZE.

Multiple-volume tape files are also supported in this release. Double buffering is always used for tape I/O processing.

7.8.1 Mount/Dismount a Tape

Tape can be mounted through the 'MOUNT' command or during 'OPEN'. The MOUNT SVC and command are described elsewhere and will not be repeated here.

The tape drive density is set to 1600 BPI for bypass-label-processing (BLP) type mount, and set to the physical tape density for normal mount.

If the tape volume is not found at OPEN, the OPEN SVC will issue a respecification message directing the user to mount the tape volume, much like the disk mount respecification. The only difference is that tape mount through OPEN is always for Exclusive use.

A tape volume mounted for SHARED use can be opened by all users. However, once the tape is opened, it will become mounted for Exclusive use and will remain so until remount or dismount.

To dismount a tape, the DISMOUNT SVC must be used or the file must be CLOSEd with Rewind-And-Unload specified.

7.8.2 Initialize a Tape Volume

The VS utility program TAPEINIT is used to initialize a new labelled volume. TAPEINIT first mounts the new tape using BLP option. The MOUNT SVC positions the tape at the Load Point Marker, and sets the Label Type to UNLABELLED. TAPEINIT then opens the NL tape in OUTPUT mode with File Sequence Number 1, and with Tape Density chosen. OPEN sets the drive density, and writes the VOL1 LABEL (in ASCII for ANSI label and EBCDIC for IBM label). At CLOSE, two tape marks are written by the CLOSE SVC to indicate that no file is on the tape. TAPEINIT then uses the Mount With No-Message option to tell the system the new tape volume name and its label type.

7.8.3 Open a Tape File

Input to OPEN SVC includes the following UFB fields:

UFBBLKSIZE (Tape Block Size):

> For NL Tape, UFBBLKSIZE is the maximum block size of the tape file.
>
> For labelled tape, VFBBLKSIZE is interpreted differently for different OPEN modes. In OUTPUT mode, it is the maximum block size of the tape file, and is the block size recorded in the tape label. In EXTEND mode for all access methods or in INPUT mode using BAM or PAM, UFBBLKSIZE is the maximum block size acceptable, unless it is set to 0, meaning that the maximum block is size unknown.
>
> Maximum size is 32K in all cases. Minimum size is 12 bytes for existing files, 18 bytes for new files.

UFBRECSIZE (Tape Record Size):

Ignored in BAM or PAM. In RAM, UFBRECSIZE is used as follows: for NL tape with fixed-length records specified, UFBRECSIZE is the record size expected. For NL tape with variable-length or undefined-length records, it is the maximum record size expected. For labelled tape (AL/IL) in OUTPUT mode, it is the record size to be recorded in the label. For labelled tape in INPUT mode, it is the maximum record size acceptable, unless it is set to 0, meaning that the maximum record size is unknown. For labelled tape in EXTEND mode, UFBRECSIZE is ignored. Tape record size must be less than or equal to BLKSIZE (no record may span two or more blocks). For RAM with fixed-length records, the UFB is considered to be in error if BLKSIZE is not a multiple of RECSIZE.

Maximum record size in 32K in all cases.

UFBFORG: Indexed files are not supported. If UFBFORGVLEN is set, OPEN will accept a variable-length record format (W-format) tape file only. If UFBFORGU is set, OPEN will accept an undefined record format (U-format) tape file only. If UFBFORGVIBM is set, OPEN will accept an IBM variable-length record format (V-format) tape file only. If none of the above flags is set, OPEN will cancel the program with an error indicator.

UFBF1: All options are supported.

UFBF2: For AL and NL tapes, INPUT, OUTPUT, and EXTEND modes are supported. For IL tapes, only INPUT and OUTPUT modes are supported. PAM in EXTEND mode is not supported.

UFBDEVCLASS: Set to X'02' to indicate tape (but see UFBF4ALLOWT, below).

UFBFLAGS: Only the 'COMPRESS' option is used.

UFBPRNAME: OPEN prname.

UFBVOLSER: Tape volume name. If the volume is not found, OPEN will either display a respecification message to request that the tape volume be mounted or take the specified OPEN Exit.

UFBFILENAME and UFBDIRNAME: The library name and file name are ignored for NL tape. For AL or IL tape, they must be specified in OUTPUT mode. In these cases, the specified names are entered into the tape label in the form 'library.file' (for example, YKW.TEST). For IL tape, translation to EBCDIC is performed. For an existing labelled file, if file sequence is supplied in UFBTSEQ, UFBFILENAME and UFBDIRNAME are nonblank and OPEN uses the library and file names to position the tape. The tape file name in the HDR1 is decoded according to the format 'library.file', and will be treated as blank otherwise. If the file sequence is supplied, OPEN positions the tape by File Sequence Number and then checks the file name (if nonblank) against the file label. File and library names in IL labels are translated to ASCII for comparison with the corresponding UFB fields.

UFBF4: UFBF4NOMSG and UFBF4ALLOWT are used. In order to open a tape file, either UFBDEVCLASS must indicate 'TAPE' (X'02) before OPEN, or UFBF4ALLOWT must be set.

UFBTLABELS: Tape label type (NL,AL,IL,ANY). If ANY is specified, OPEN will allow any label type tape file. If more than one type is specified, OPEN will accept tape files with the specified label types.

UFBTDEN: Tape density (800 BPI, 1600 BPI). If UFBTDEN is set to binary zeroes (tape density not specified), OPEN will set the UFBTDEN to the density found in the UCB. If UFBTDEN is specified, and the tape volume is not mounted for "Bypass Label Processing" (BLP), OPEN will check the UFBTDEN against the actual tape density, and will respecify or take the OPEN Exit if they do agree. If BLP is in use, the tape drive density will be set to the density according to UFBTDEN, and will be recorded in the UCB.

UFBTSEQ: File sequence number. OPEN uses this number to position the tape. It must be specified for NL tape and for labelled tape in OUTPUT mode. For existing files on a labelled tape, one can also position by file name.

OPEN EXIT FLAGS: Tape OPEN Exits are handled exactly as disk OPEN Exits. The OPEN Exit masks are set up in the first byte on the top of the stack in the SVC input parameter. Currently supported OPEN Exits are:

(1) Volume not found
(2) File not found, or sequence number out of range
(3) No space available on tape volume to create new file
(4) Possession conflict
(5) Wrong label type or wrong density

No other fields in the UFB are used as 'OPEN' input.

Output of the OPEN SVC includes the following UFB fields:

UFBVREAD and UFBVSTART: Set to DMS routine entry address.

UFBFS1 and UFBFS2: Set to OPEN Return Code. If UFBF4NOMSG is requested and OPEN is not successful, a 4-byte message ID is set at the beginning of the UFB, just as for Disk OPEN. If OPEN Exits are set, UFBFS2 is the exit flag. (The exit flag indicates which OPEN Exit condition is taken, just as for Disk OPEN Exit.)

UFBBLKSIZE: For labelled tape, existing file (INPUT and EXTEND mode), UFBBLKSIZE is set to the tape block size from the file label. This is the maximum allowable block size for DMS operations.

UFBRECSIZE: For an existing file on labelled tape, (INPUT and EXTEND modes), using RAM, UFBRECSIZE is set to the record size from the file label, and is the maximum acceptable record size for DMS operations. For an existing file on labelled tape, using RAM with undefined record format, it is set to the block size from the tape label, and is the maximum acceptable record size for DMS.

UFBFORG: Set to actual tape file organization in INPUT and EXTEND modes.

UFBFLAGS: Compress flag set for an "X" format file in INPUT and EXTEND modes.

UFBF1: UFBF1PREVO and UFBF1OPEN flags are set.

UFBDEVADDR: Set to the device number of the device on which the tape volume is mounted.

UFBFILENAME and UFBDIRNAME: For a labelled tape in INPUT and EXTEND modes, OPEN will decode the tape file name in the HDR1 and put it in these fields. The name in the label must be in the form 'library.file' in order to be decoded. Otherwise, these fields are left blank.

UFBNRECS: Set to 0.

UFBLRECSAVE: Set to the same value as that in UFBRECSIZE.

UFBTLABELS: Will be set to the actual tape label type after successful OPEN.

UFBTDEN: Will be set to the actual tape density after successful OPEN.

UFBTSEQ: Will be set to the tape file sequence number after successful OPEN.

UFB fields in the DMS section are initialized. No other fields are used.

OPEN SVC functions - Existing File (INPUT and EXTEND Modes):

OPEN issues a 'GETPARM' to obtain the volume name, file name, and file sequence number. If the volume is not mounted, a message is displayed requesting the user to mount the volume. The tape is then positioned by file sequence number or file name. Information in the file label (for labelled tape) is then extracted and placed in UFB fields and control blocks are allocated. UFB fields are initialized and the tape positioned to the first data block for INPUT mode to the tape mark following the last data block for EXTEND mode. Two I/O buffers of the size specified in UFBBLKSIZE are allocated in the Segment 2 buffer area.

OPEN SVC Functions - New File (OUTPUT MODE):

OPEN issues a 'GETPARM' to obtain the volume name, file name, and file sequence number. If the volume is not mounted, a message is displayed requesting the user to mount the volume. The tape is then positioned by file sequence number. For labelled tape, the file labels HDR1 and HDR2 are constructed and written on the tape (and followed by a tape mark). Library and file names are converted to the form (lib.file) and placed in the HDR1 file. (No labels are written for NL tape.) Control blocks are allocated. UFB fields are initialized, and the tape is positioned to the first data block. Two buffers are allocated (as for INPUT mode).

7.8.4 READ Function Request

INPUTS: The following UFB fields are used:

UFBEODAD,
UFBERRAD: Error Exit address.

UFBVREAD: Function modifier.

UFBRECAREA: Record area for the record read.

Outputs: The following UFB fields are set:

UFBFS1,
UFBFS2: File status after DMS functions.

UFBRECSIZE: Set to record size after READ in RAM.

Set to block size after READ in BAM. For "U" format file, in RAM, RECSIZE is the block size of the block which was read.

UFBNRECS: Number of records processed.

DMS Routine Operations: The next record on the tape file is moved to user-specified record area. For multiple-volume tape files, volume switching may be performed (see 7.8.8 below). If the NODATA option is used, the record is not moved to the record area. Instead, the record pointer is returned in general register 1.

7.8.5 WRITE Function-Request

Inputs: The following UFB fields are used:

UFBEODAD and UFBERRAD: Error Exit address.

UFBRECAREA: Record area of the record to be written.

UFBBLKSIZE: Tape block size to be written. (Used in PAM only.)

UFBRECSIZE: Record size, specified when writing records for variable-length-record files (record formats V, W, X, and U).

Outputs: The following UFB fields are set:

UFBFS1, UFBFS2: File status after DMS functions.

UFBNRECS: Number of records processed.

DMS Routine Operations: The record addressed by UFBRECAREA is written to the tape. For multiple-volume tape files, volume switching may be performed (see 7.8.8 below).

7.8.6 START Function-Request

Inputs: The following UFB fields are used:

UFBEODAD, UFBERRAD: Error Exit address.

Outputs: The following UFB fields are set:

UFBFS1, UFBFS2: File status after DMS functions.

DMS Routine Operations: To wait for the completion of a previous READ/WRITE request (in PAM only).

7.8.7 Close Tape File

Inputs: UFB fields used by CLOSE:

All UFB fields left from previous DMS functions will be used by CLOSE. No unauthorized modifications should be made to the UFB. Options supported are "NOREWIND', 'UNLOAD', and 'REEL'.

Output: UFB fields used by CLOSE:

UFBFS1,
UFBFS2: Return Code.

UFBF1: UFBF1OPEN flag turned off

CLOSE SVC Functions: In OUTPUT and EXTEND modes, CLOSE writes out data in buffers. For labelled tape, the trailer labels EOF1 and EOF2 are constructed and written to tape, followed by two tape marks. For NL tape, only two tape marks are written; trailer labels are written. In INPUT Mode, CLOSE completes the last I/O operation. CLOSE then deallocates control blocks and buffers, cleans up UFB fields, rewinds the tape, and returns unless the 'NOREWIND' or 'UNLOAD' options are specified. For the 'NOREWIND' option, CLOSE positions the tape to the tape mark following the processed tape file, rather than rewinding it. For the 'UNLOAD' option, the tape is rewound, unloaded, and logically dismounted.

The 'REEL' option asks CLOSE to terminate the current volume with an 'End Of Volume' label and continue the tape file on next volume (see 7.8.8 Multiple-Volume Tape File, below).

7.8.8 Multiple-Volume Tape File

The multiple volume tape processing is supported by DMS for AL and IL tape files, as described below.

To Read a Multiple-Volume Tape File (INPUT Mode)

When end-of-tape is reached, DMS reads the trailer label (if neither EOF1 nor EOV1 is found, a warning message will be displayed, and the End-Of-Data exit will be taken), compares the tape block count with the actual number of tape blocks read, and warns the user if the counts are inconsistent.

If the trailer label is an EOF1 block (indicating end-of-file reached), the End-Of-Data exit is taken with File-Status = 10.

If the trailer label is an EOV1 block (indicating file to be continued to next volume), and if the user has specified the NOMSG option or EOD=EOV option, the End-of-Data Exit is taken with File-Status = 11.

Otherwise, the tape file is continued to the next volume. A volume switch message is displayed, directing the user to mount the next volume. The newly mounted volume is checked for the correct volume sequence number and the 'READ' operation continues from the new volume.

To Create a Multiple-Volume Tape File (OUTPUT Mode)

When end-of-tape is reached, DMS checks the NOMSG flag and EOD = EOV flag. If either of these flags is set, the Boundary Violation Exit is taken with File-Status = 34. Otherwise, the EOV1 and EOV2 blocks are written on the tape followed by two tape marks, and DMS displays a message asking the user to specify the next volume and device. A MOUNT SVC is issued asking the user to mount the next volume and the WRITE operation continues on the next volume.

7.8.9 7-Track Tape Support

The 7-track tape records only six bits of data for each byte. Thus, only the six low-order bits of each byte are written on tape and, during a read, the two high-order bits are set to zeros in main memory.

Data can be recorded on the tape using odd or even parity. Note, however, that a data pattern of X'00' cannot be recorded on tape with even parity; instead, it is recorded as X'0A'. If it is recorded with odd parity, it is recorded as X'00'.

Data Management supports only unlabelled tape files for 7-track tape. The tape mount function is the same as for 9-track tape; tape density is set to 800 bpi (the only density supported). System control blocks always indicate label type 'NL'.

When opening a 7-track tape file, the user must specify the parity of the file in UFBTPARITY, even for an existing file. All other UFB fields are used as for 9-track tape. In RAM, only fixed-length and undefined-length records can be used for 7-track tape. BAM and PAM are supported as for 9-track tape.

All DMS functions as well as CLOSE are then same as four 9-track tape.

7.9 PHYSICAL ACCESS METHOD FUNCTIONS

Any disk file can be accessed under the Physical Access Method (PAM) when UFBF1PAM is set in the UFB (at OPEN.) In this case, the file is the accessed by block number (from zero) rather than record number. SVC OPEN does not allocate I/O buffers when UFBF1PAM is set. All I/O is performed by reading or writing directly from a user-supplied address. The length of the data transfer is also set by the user-program.

NOTE: The Physical Access Method (PAM) provides the user with asynchronous waits for I/O operation completions. A physical I/O operation is initiated by a read, rewrite or write function request. The data area for the physical I/O operation is defined by UFBRECAREA and the data length (UFBBLKSIZE) as returned in the UFB by the DMS routine. This data area should not be used until the I/O operation has been completed successfully; ie., it should not be used until a wait for I/O completion (START(WAIT) function-request) has been executed.

7.9.1 UFB Field Definitions for Physical Access Method

1. UFBMAXTFR is a 2-byte positive number which indicates the maximum data length which may be transferred per I/O request on a particular disk drive. This field is available after OPEN and will be a multiple of the disk block size (2048 bytes). UFBMAXTFR is 2048 if the device is a printer.

2. Common PAM Input Parameters

   a. UFBERRAD, UFBLF, UFBLFMOD and the UFB BCBsection are used as in the Common UFB Input Parameters. UFBEODAD is taken when File status = '23' (record not found) is returned.

   b. UFBKEYAREA - Pointer to word containing the beginning block number of the data to be read or written. The first block in the file is block number zero.

   c. UFBRECAREA - Address (Segment 2) of data. Data areas are completely controlled by the user-program. The data address must be properly aligned; otherwise, file-status = '96' will be set for invalid command. (Page alignment is required).

d. UFBBLKSIZE - This 2-byte field indicates the length for the data transfer desired. DMS function-requests will set UFBBLKSIZE equal to the length of the data actually read or written when end-of-extent, end-of-cylinder, or end-of-file cause the data transfer to be truncated. UFBBLKSIZE must be a multiple of 2K or file status = '96' will be set for invalid command.

UFBBLKSIZE may equal UFBRECSIZE = 2K for appropriate files. In this way, files with block size = record size may be accessed without record movement and without any extra user-program support. In this case, UFBBLKSIZE will never be reset by DMS (i.e., the data transfered will always = one disk block.)

7.9.2 Read Block Function-Request (INPUT or I/O)

Inputs: a. Common PAM Input Parameters

b. UFBEBLK - block number of last block within file

DMS routine operations: The desired block(s) as indicated by UFBKEYAREA and UFBBLKSIZE is read into the user area (UFBRECAREA). The read block function is random (block number required) and asynchronous. (START is used to wait for completion.) File status is set equal '95' if there is I/O in progress (invalid function sequence). File status is set equal to '23' (record not found) if the block number supplied is greater than UFBEBLK or if the file contains no records (UFBEREC=0 and UFBEBLK=0). UFBEODAD is taken for file status = '23'.

Outputs: UFBFS1, UFBFS2 stored (operation initiated ok or errors as above).
BCB updated for read operation initiated.
UFBLF set for read.
UFBBLKSIZE set equal to actual length of the transfer.

Note: UFBEREC and UFBRECSIZE are available in the UFB and may be used by the user-program to determine the last record in the last block (end-of-file). (UFBEREC equals the number of records in the last block of the file; UFBEREC=0 only when the file contains zero records (null-file.)

7.9.3 Rewrite Block Function-Request (I/O)

Inputs: a. Common PAM Input Parameters

b. UFBEBLK - block number of last block within file.

DMS routine operations: The desired block(s) as indicated by UFBKEYAREA and UFBBLKSIZE is written from the user area (UFBRECAREA) to the disk file. The rewrite block function-request is random and asynchronous (see "Read Block", Subsection 4.9.1). The rewrite block function-request does not require a prior read block operation. File status errors ('23' or '95') may occur (as in Read block). UFBBUFSIZE is set following the initiation of the write operation. (No extents are allocated in I/O mode.)

Output: UFBFS1, UFBFS2 stored
BCB updated for write operation initiated.
UFBLF set for REWRITE
UFBBLKSIZE set equal to the actual length of the data transfer

7.9.4 START Function-Request (INPUT, I/O, or OUTPUT)

Inputs:
a. Common PAM Input Parameters
b. UFBEBLK - block number of last block within file
c. Modifier byte at UFBVSTART

DMS routine operations: This function-request has two different operations depending on the value of the modifier byte:

START (WAIT) - The last I/O operation is waited on for I/O completion. File status = '95' if there is no I/O in progress. The file status of the completed I/O operation is available in UFBFS1, UFBFS2 after START (WAIT); the record area is also available to the user-program after I/O completion.

The START (WAIT) function-request is required for waiting on I/O completion for all the asynchronous PAM operations.

START OUTPUT/START IO - This request is available only in OUTPUT mode and in temporary I/O mode. It is similar to the START request for consecutive disk files (Subsection 4.4.4)

START IO - The last I/O operation (if any) is waited on. Then the vectors are modified to provide the Read block and Rewrite block function-requests. The size of the file is determined by UFBEBLK.

START OUTPUT - The last I/O operation (if any) is waited on. Then the vectors are modified to provide the write block function-request. UFBEBLK is set equal zero for no records in file.

START EXTEND - This option is similar to START OUTPUT except that UFBEBLK is adjusted so that subsequent WRITES will add blocks to the end of the file.

Outputs (START OUTPUT and START I/O):

UFBFS1, UFBFS2 stored
BCB section updated (no I/O in progress)

For Start I/O- UFBVWRITE set for invalid functions; UFBVREAD and UFBVREWRITE set for I/O mode functions.

For Start Output - UFBVREAD and UFBVREWRITE set for invalid function. UFBVWRITE set for OUTPUT mode write function.

UFBFS1, UFBFS2 stored in the UFB; UFBLF and UFBLFMOD set.
BCB section updated for no I/O in progress.

7.9.5 Write block function-request (OUTPUT)

Inputs: a. Common PAM Input Parameters
b. UFBEBLK - block number of last block within file

DMS routine operations: The desired block(s) as indicated by UFBBUFBLOCK and UFBBLKSIZE are written from the user area (UFBRECAREA) to the disk file. UFBBUFBLOCK is the number of the next highest block beyond the current end of the file. UFBBUFBLOCK is maintained by DMS and allows a file to be created sequentially; it should not be modified by the user program. The write block function-request is thus sequential and asynchronous. The file status for Invalid function ('95') occurs if the last write has not been waited on. UFBBLKSIZE is set to indicate the length of the data transferred after I/O initiation. Also, UFBEBLK and UFBBUFBLOCK are updated for the number of blocks transferred. Additional extents are added as necessary; file status = '34' when an additional extent is required and none can be allocated.

Outputs: UFBFS1, UFBFS2 stored
UFBBLKSIZE set equal to the actual length of the data transfer
UFBBUFBLOCK and UFBEBLK updated for number of blocks tansferred.
BCB section updated (write initiated)
UFBLF set for write.

APPENDIX A - DATA AREA MACROINSTRUCTION FORMAT

Corresponding to each system data structure is a macroinstruction which may be used freely by system and user's programs to define standard labels for fields within the structure. If only the macroinstruction name is written, a dummy section (DSECT) of that name is generated. If a register specification (e.g., UFB REG=R1) is included, a USING pseudo-operation is also generated (e.g., USING UFB,R1). If the "SUFFIX=" operand is provided, each label is generated to contain the SUFFIX character immediately following the block name (e.g., for "UFB SUFFIX=#", the label "UFBF1" becomes "UFB#F1"). Only a one-character suffix should be specified. If "NODSECT" is specified (e.g., "UFB NODSECT, REG=R1"), the DSECT (and termination CSECT) pseudo-operations are not generated. The block name is not included as a label name in this case. The general format of these macroinstructions is as follows:

```
                    MACRO
                    blockname  &NDS,&REG=,&SUFFIX=
                    GBLB       &NODS,&blockname
                    AIF        ('&SUFFIX' NE'').L0
                    AIF        (&blockname).L3
&blockname          SETB       (1)
.L0                 ANOP
                    AIF        ('&NDS' EQ 'NODSECT' OR &NODS).L1
blockname&SUFFIX    DSECT
.L1                 ANOP
*
*      BLOCK DEFINITION FITS IN   HERE
*
                    AIF        ('&NDS' EQ 'NODSECT' OR &NODS).L2
&SYSECT             &SYSTYP
.L2                 ANOP
                    AIF        ('&REG' EQ '').L3
                    USING      blockname&SUFFIX,&REG
.L3                 ANOP
                    MEND
```

APPENDIX B - USER PROGRAMS IN THE WANG 2200VS

THE PROGRAM

A program is an entity invoked by the command processor or by a LINK SVC. The name of the program is the two-level name of its disk file. There is only one program in a file. A program file is a consecutive file with 1K (1024) byte records and the 'program' flag set in its file descriptor record. The object program is originally built by a language translator (compiler or assembler). It then may be modified by a linker program.

THE PROGRAM SKELETON

A program is partitioned into blocks of information. Each block represents common information that will be needed either when the program is running or when the program is processed by the linker program. To describe the skeleton at this level and for the lower levels, a special notation will be used. Each item to be described will be listed in a table. To the left of the item's description will be its 'level' number and its length. The level number is a designation of the relationship of this item with the preceding items in the table. The start of the table will be the program with a level number of zero. Then to describe structures within the program, there will be one or more level one items listed under the level zero item. The first of the items with the next higher level number will be assumed to be located at the start of the item specified with the lower level number. To help see how this system works, take a look at the following example:

Level Number	Length	Description
0	var	A
1	5	B
2	3	C
2	2	D
1	var	E

Example 1

In this example, the whole is 'A'. The first level below 'A' is level 1. There are two parts to this level: 'B' and 'E'. The start of B corresponds to the start of A. At the next level, level number 2, there are also two parts. These are 'C' and 'D'. The start of C corresponds to the start of B and also the start of A. A slightly more general use of this notation would specify that there can be any number of replications of B. If this was the case:

Level	Length	Description
0	var	Object program
1	var	The RUN block
1	var	The SYMBOLIC block
1	var	The LINKAGE block

THE RUN BLOCK

This contains the information needed by the system to run a program. The block will be used by the operating system as a paging file when the program is running. It contains the actual instructions to be run and the information needed to format the 'static' area on the stack when the program is started.

Level	Length	Description
1	var	The RUN block (First location has an assumed address of 1,048,576 (or 1024K).)
2	var	CODE and PROLOG block
3	8	PROLOG block
4	4	Length of CODE and PROLOG block (This value is used to find the start of the 'STATIC' AND LENGTHS block.)
4	4	Entry point address This is the point to which control will be passed when the program is started. It will be the address of any external name in the CODE sections. If the high-order bit is 1, the program has been assembled to run in segment 0 (as for standalone utilities and operating system routines).
3	var	CODE block This may contain any number of CODE sections. This block contains all of the executable instructions in the program and may contain nonmodified data. It is composed of any number of sections, where a section is contiguous area of code that can be moved by the linker program as whole. There is no requirement for a particular order of the sections within the CODE block.

Level	Length	Description
4	var	CODE section This is a block externally identified by its name. It is an independent contiguous area of code supplied by the language translator. The first location will be on a double word boundary, and the length will be divisible by eight. All address constants that are resolvable will be resolved so that the program can be run without changing any locations in the section.
2	16	LENGTHS block
3	4	Length of 'static' block in bytes (object time). This length reflects the length of data in the STATIC area. If the length is not divisible by 4, up to four bytes of slack will be added after the end of the block to make the following block start on a word boundary. These slack bytes are not counted in the length.
3	4	Length of 'static' area in bytes (run time).
3	4	Reserved, must be zero
3	4	Reserved, must be zero
2	var	'Static' block This block contains sections of initial value records. There can be any number of 'static' sections in this block (including zero).

'STATIC' BLOCK

This block contains initial value records that are to be processed by the program startup facility in the operating system. These records will cause initial values to be assigned to locations in the 'static' area. There can be any number of 'static' sections within this block. All address constants in the static sections that reference locations in the CODE sections will be resolved by linker or translator programs as if they were in a code section. Address constants that address locations within the 'static' sections will be resolved as if the start of the 'static' block were location zero.

Level	Length	Description
2	var	'Static' block
		This may contain any number of 'static' sections (including zero).
3	var	'Static' section
		A 'static' section can be any length, including zero. The section contains only the compressed initial value records for this section. If no locations in the static section are to have initial values, there will be no records for that section and the object time length will be zero. It should be noted that the length of these 'static' sections does not correspond to the length of the expanded 'static' section at run time. In order to distinguish between the two, the following naming convention will be used. Locations in the object code (the disk file) will be referred to as object time locations or will be specified by their object time address. Locations that are used during running of a program will be referred to as run time locations or will be specified by their run time addresses. Because the CODE block is used without change at run time, this distinction will normally not be made for any locations of the code block. When descriptions apply to both static and code areas, 'run time' and 'object time' may be used interchangeably to refer to the code area.

Level	Length	Description
4	5-2054	Initial value records These records specify locations that are to have initial values in the named 'static' section and the values the program startup mechanism is to assign to these locations. There are five types of initial value record: 1) The origin record, which specifies how far from the start of the expanded run time static area this section will start. 2) The value record which specifies the value to be placed into the static area. 3) The relocation record which specifies that program startup is to supply the address of a run time location in a static area. 4) The repeated record which specifies a value and a repetition factor to indicate how many occurrences of the value are to be placed in the static area. 5) The compressed record which specifies the compressed value to be expanded and placed into the static area.
5	1	Length of data within the data field in this record minus one (this field is not used for the compressed record type).
5	.4	Record type. = 0 Value = 1 Origin = 2 Relocation = 4 Repeated = 8 Compressed
5	2	Run time displacement This field has two interpretations. For origin records (record type - 1), this indicates the displacement from the start of the static area of this static section. For all other record types, this indicates the run time displacement from the start of the static section of the record's data.

5	1-256	Data field. The length and format of this field varies depending on the record type as follows:

Data field of value type record

Level	Length	Description
5	1-256	Data to be moved to the 'static' section. This data is moved unchanged to the run time static area.

Data field of origin type record

Level	Length	Description
5	1	One byte of dummy data.

Data field of the relocation type record

Level	Length	Description
5	4	Relocation item These are similar to the relocation entries in the linkage block of the program. Each one represents an address constant in the static section and the initial value the location is to have. Every address constant in a STATIC section must be initialized by a relocation item.
6	.3	Reserved, must be zero
6	.1	Base for relocation = 0 Relocate relative to the static area = 1 Relocate relative to the current static section
6	.1	Length of target address constant = 0 Three bytes = 1 Four bytes
6	.1	Direction of relocation = 0 Positive = 1 Negative

6	.1	Do not relocate flag = 0 Relocate the address constant = 1 Move the address constant to the specified location but do not relocate with respect to the static area. This flag is used if the address is unresolved, if the address referenced a code location, or if this is an R type address constant.
6	.25	Initial value of the address constant If the target is three bytes long, only the last three bytes will be moved to the target area and the high-order bit will be ignored. If the target is four bytes long, the high-order bit will be propagated through the seven remaining bits of the high-order target byte.

Data field of repeated type record

Level	Length	Description
5	2	Repetition factor (2-32767)
5	1-256	Data to be repeated wthin the static area.

Data field of compressed type record

Level	Length	Description
5	2	Length of compressed data. (1-2048)
5	1-2048	Compressed data This data is expanded before being moved to the static area.

THE SYMBOLIC BLOCK

The symbolic block is a pool of information to be used by the system's debugging support. The block is partitioned into a length field and any number of section related blocks.

Level	Length	Description
1	var	The SYMBOLIC block This contains all of the program's special debugging information.
2	4	Fullword Aligned Length of SYMBOLIC block. (If there are no symbolic sections, the length will be 4.)
2	var	Section area This can contain any number of SYMBOLIC sections.
3	var	SYMBOLIC section Every CODE section can have a SYMBOLIC section. If it does, these sections will be in the same order as the corresponding CODE sections.

THE SYMBOLIC SECTIONS

The SYMBOLIC section contains information used by the system's debugging support for the named code section. It is partitioned into a part at the start of the block that contains all of the external labels that are referenced, followed by any number of subblocks of specialized debugging information. Each of these subblocks will contain information grouped by type (i.e., one of the blocks may be the 'statement' number block with the list of locations of each line of source code's generated object code).

Level	Length	Description
3	var	The SYMBOLIC section. This is the pool of debugging information for the system debugging support for the named section.
4	4	Length in bytes of data in this symbolic section (including this word). This length reflects the length of data in the area. If the length is not divisible by 4, up to three bytes of x'00' filler will be added after the end of the section to make the following section start on a word boundary. These slack bytes are not counted in the length.
4	8	External name of corresponding CODE section.
4	4	Doubleword aligned object time length of the corresponding section block in the RUN block.
4	22	Location of source file at compilation time.
5	8	Source filename.
5	8	Source Library Name.
5	6	Source Volume.

Level	Length	Description
4	22	Location of listing file at compilation time (or blank).
5	8	Listing filename.
5	8	Listing Library name.
5	6	Listing Volume.
4	var	External reference pool. This lists all labels that are externally referenced. Each label is listed only once, no matter how many times it is used in the program. There is no order in the list, but the position of an entry in the list represents the internal number used for referencing that label. This structure allows modules to be added or dropped by a linker program without changing any locations in SYMBOLIC section (other than adding or dropping the whole section).
5	4	Length of external reference pool (in bytes), including this word.
5	var	Any number of external Reference Entries.
6	12	External Reference Entry.
7	8	External name (ASCII with trailing blanks).
7	4	Displacement within code section to 4-byte RCON.

Level	Length	Description
4	var	SYMBOLIC subblock area. This may contain any number of subblocks. Each subblock is composed only of one type of debugging information (i.e., 'statement number' block).
5	var	SYMBOLIC subblock.
6	1	Subblock type. The types are language independent codes and are interpreted the same for all languages.
6	3	Length of data in subblock. This length reflects the length of data in the subblock. This includes the first four bytes of the subblock. If this length is not divisible by 4, up to 3 bytes of X'00' filler will be added at the end of the subblock to make the following subblock start on a fullword boundary. These filler bytes are not counted in the length.
6	var	Collected information about this section. These subblocks are processed by a common language independent program.

STATEMENT NUMBER SUBBLOCK

This subblock is generated by all high-level language translators. It contains language independent information identifying individual statements in the program's section. Exactly one statement number subblock must be present in each symbolic section.

Level	Length	Description
5	var	STATEMENT NUMBER subblock.
6	1	= 1 (STATEMENT NUMBER type subblock).
6	3	Length of data in the subblock including this word. If this length is not divisible by 4, up to 3 bytes of X'00' filler will be added to the end of the subblock to make the following subblock start on a fullword boundary.
6	var	Any number of statement entries. Each entry represents one 'statement' in the source program. The definition of statement is language dependent, but is consistent within any one language (i.e., in COBOL there will be one entry per verb in the COBOL source). The entries will be in order of increasing displacements.
7	10	Statement entry.
8	2	Line number in binary (no negative values) or zero to indicate inline nonsymbolic code. The exact definition of this entry is language dependent, but normally will indicate the statement line number.
8	5	Character string (in ASCII with trailing blanks), or XL5'00' to indicate COBOL, perform paragraph header linkage. The use of this field is language dependent, but can be used either for the statement label or the command starting at the specified displacement. (In COBOL this will be used for an abbreviation of the COBOL verb.)

Level	Length	Description
8	3	Displacement (run-time) into the code SECTION. This is the run-time displacement into the section of the start of the statement.

DATANAME SUBBLOCK

This subblock is generated by all high-level language translators to support symbolic access to data items at runtime through command language facilities. Exactly one dataname subblock must be present in each symbolic section.

Level	Length	Description
5	var	DATANAME SUBBLOCK
6	1	=2 (Data name type subblock).
6	3	Length of data in subblock (including this word). If this length is not divisible by 4, up to 3 bytes of X'00' filler will be added at the end of the subblock to make the next section start on a fullword boundary,
6	var	DATANAME INDEX This index contains compiler dependent information used to efficiently search the subblock or a given symbol.
7	4	Number of index items.
7	var	Any number of positional index items.
8	4	Index item = displacement within symbolic section to first Dataname Entry with the compiler dependent indexed attribute. (For COBOL and BASIC, the index is by alphabetical order and points to groups of Dataname Entries for symbols beginning with the same character. This implies that there are exactly 26 Dataname Index entries for COBOL and BASIC.)
6	var	Any number of data name entries. These can be in any order. It is expected that any one compiler will order these such that when the compiler is identified they can be efficiently searched.

Level	Length	Description
7	var	Data name entry.
8	1	Length of entry minus one.
8	5	Path to the data.
9	1	Index of the external section that the displacement references. This is the number of the entry in the external reference pool in the symbolic section.
9	1	Type of path to the data item; 0 = Displacement locates the data constant in the corresponding code section 1 = Displacement locates the data item in the references external section. 2 = Displacement locates a four-byte ACON in the referenced external section which should be used as a base address. The displacement from this address is found in the offset field. 3 = Displacement equals value (right-justified)
9	3	Displacement from the indexed external section to the data item (for type 2 path this is the displacement to the address constant).
8	3	Data description.
9	1	Type of variable.
10	.1	Indicator - referenced item (whether referenced using subscripts or not) is an elementary data item if = 1.
10	.1	Indicator - Subscripts required if = 1.

Level	Length	Description
10	.6	Format Indicator;

0 = Mixed (applies to nonelementary items only and implies that the scale is to be used as the high-order byte of the variable-length field).

1 = Character (this implies that the scale is to be used as part of the variable length).

2 = Binary.

3 = Packed decimal.

4 = Bit string (value always interpreted as full bytes).

5 = Floating point.

6 = Display Field Attribute Character

8 = Zoned number with no high or low order sign zones but may contain either a leading or trailing sign character and one deciaml point character.

9 = Binary COBOL halfword index value (length per occurrence number is in item length - length is 2 bytes).

10 = Zoned Numeric with High-Order Sign Zone.

11 = Zoned Numeric with Low-Order Sign Zone.

12 = Zoned Numeric with Leading Sign Character.

13 = Zoned Numeric with Trailing Sign Character

Level	Length	Description
		Format Indicator:
		14 = Unsigned Zone Numeric
		Note: Zoned numeric fields contain all numeric characters, 0-9, except for sign. Scale factor indicates decimal point location.
		15 = BASIC array.
		16 = COBOL group item.
		17 = BASIC string scalar.
		18 = Binary COBOL fullword index value (Length per occurrence number is in item length. Length is two bytes).
		19 = Logical (FORTRAN)
		20 = Complex (FORTRAN)
9	1	Scale
		This is a signed binary number indicating how far left of the rightmost digit the decimal point is to be relocated (relocation is to the rigth fro negative numbers). (For character type 1 fields, this byte will be considered part of the item length.)
9	1	Data item length.
		This specified the length of the data time. If the data is character type, both this and the preceding byte will be used for the length.
8	1	Length of data name minus one.

Level	Length	Description
8	var	Data name. If the data name must be qualified, all necessary levels of the name are to be listed with highest level first and the levels of qualifications separated by points ('.'). If the name is qualified but is unique in the program, only the lowest level of the name should be listed.
8	2	Optional offset (present with data path 2 only).
8	var	Array descriptor.
9	1	Number of subscripts required. There will be one dimension description for each subscript.
9	var	Indicated number of dimension descriptions.
10	4	Dimension description. There will be one entry for each subscript indicated in the maximum subscript value (entry for leftmost subscript first). The first element of each dimension of the array will be assumed to be 1.
11	2	High bound of subscript
11	2	Length of Subscript Item.*

* For the general array: $A[1...u_1,1...u_2,...,1...u_n]$, the "Length of Subscript Item" for the jth subscript (LSI_j) is defined as follows:

For Row Major Ordering,

$$LSI_j = \begin{cases} \prod_{k=j+1}^{n} u_k & 1 \le j \le n-1 \\ 1 & j=n \end{cases}$$

For Column Major Ordering, LSI_j =

$$\begin{cases} 1 & j=1 \\ \prod_{k=1}^{j-1} u_k & 2 \le j \le n \end{cases}$$

The Length of Subscript Item entry is used by internal routines of the Symbolic Debugger to efficiently locate array elements in a storage independent manner. The following formula is used to locate element $A[i_1,i_2.....,i_n]$:

$$LOC\ A[i_1,i_2.....,i_n] = LOC\ A[11.....1_n] + \sum_{j=1}^{n} LSI_j*(i_j-1)$$

THE LINKAGE BLOCK

This is a pool of information required for the linker program to add or delete sections of the program. It is partitioned into a length field and then any number blocks of section information.

Level	Length	Description
1	var	The LINKAGE block.
2	4	Fullword aligned length of the LINKAGE block.
2	var	CODE section block area. This is composed of exactly one section block for each CODE section. The blocks are in the order in which the CODE sections appear in the CODE block of the RUN block.
3	var	CODE section block.
2	var	'Static' section block area. This is composed of exactly one section block for each 'static' section. The blocks are in the order in which the 'static' sections appear in the 'STATIC' block.
3	var	'STATIC' section block.

CODE AND 'STATIC' SECTION BLOCKS IN THE LINKAGE AREA

These blocks have sufficient information so that sections can be added to or deleted from a program, and so that all addresses can be correctly resolved after this operation is complete. With minor exceptions both the CODE and the STATIC blocks have the same basic skeleton. One skeleton will be presented, and the differences will be noted in the skeleton.

Level	Length	Description
3	var	Section block (CODE or STATIC).
4	4	Length of data in this section block in bytes.

This length reflects the length of data in the area. If the length is not divisible by 4, up to three bytes of X'00' filler will be added after the end of the block to make the following block start on a word boundary. These filler bytes are not counted in the length.

4	8	External name of section (in ASCII with trailing blanks).
4	1	Type of block: = 0 for code and = 1 for static.
4	3	Doubleword aligned object time length of the corresponding section block in the RUN block.
4	4	Compiler name or designation (ASCII with trailing blanks).
4	1	Version and modification level of the compiler (packed decimal) VM.
4	3	Date of compilation of this section (packed decimal) YYDDD.

Level	Length	Description
4	4	For a CODE section block: Fullword aligned length of the corresponding SYMBOLIC section (or =0 if there is no corresponding SYMBOLIC section). For a STATIC section block: Doubleword aligned run time length of section block in the RUN block.
4	var	ENTRY POINT list. (This is a list of all names in this section that are known outside of the section.) This list may have any number of names.
5	4	Length of ENTRY POINT list (including this word).
5	var	The list. This may contain any number of names and address pairs. They may be in any order, but must not be repeated.
6	8	ENTRY POINT name (ASCII with trailing blanks).
6	4	Runtime displacement into the section of this ENTRY POINT.
4	var	RELOCATION REFERENCE BLOCK. This lists all locations within the section that will need to have addresses changed if the relative location of the section within the program is changed or if the location of specified external labels changes.

RELOCATION REFERENCE BLOCK

All address constants in the section that would be relocated if either the starting location of the section or the location of an external name was changed will be listed in this block. The block is composed of two main parts: a list of external names and a list of addresses in the section to be relocated.

Level	Length	Description
4	var	RELOCATION REFERENCE BLOCK. This block contains all of the information required to relocate address constants in this section.
5	4	Actual length in bytes of the RELOCATION REFERENCE BLOCK. This includes this word, the EXTERNAL NAMES BLOCK, and the relocation items.
5	var	EXTERNAL NAMES BLOCK. This block contains a list of all external names referenced by this section. An external reference is an address constant that references a label that is not part of the current section. The label must be the name of a section or an ENTRY type symbol in a section.
6	4	Length of external names block (including this word).
6	var	List of external reference names. These names can be in any order, but they will be referenced by their position in the list. The first name is number one.
7	8	External name (ASCII with trailing blanks).
5	var	List of relocation items. This list is in order of increasing displacements. There can be any number of entries in this list.

Level	Length	Description
6	5	Relocation item.
7	1	Flag byte.
8	.2	Reserved, must be zero.
8	.1	RCON if = 1. The referenced name must be in a STATIC section. If the address constant is in a STATIC section, the relocation record will have the do not relocate bit set.
8	.1	Address is a relocation record (8 bytes). (If this is set, the following bit of length of target is ignored.) All items in a 'STATIC' section except origin records and none in a CODE section will have this bit set.
8	.1	Length of target address constant: = 0 Three bytes. = 1 Four bytes.
8	.1	Direction of relocation: = 0 Positive (add the address of the start of the section to the specified location. = 1 Negative (subtract from the location).
8	.1	Unresolved flag if set to =1 (if set, the address will be resolved relative to an address of X'F00000').
8	.1	Reserved, must be zero.
7	3	Object time displacement into the section of the address constant.
7	1	Number of external name referenced. This number is either zero if the item is to be relocated relative to the start of this section or is the number of the external name in the EXTERNAL NAMES BLOCK (the first name in the list is one).

TRANSLATOR PROCESSING

The object program is produced by a language translator. At this time, all address constants will be resolved, or marked as unresolved. The program can then optionally be processed by the linker program. It will add, rearrange or delete sections in the program. After either a language translator has produced an object program or a linker program has processed it, the program can be invoked by the operating system.

The language translator creates a complete object program that is runnable by the operating system (if no sections are unresolved at translation time). This program must have linkage information for the linker program to resolve addresses after adding or deleting sections. The translator also has the responsibility to generate the symbolic section of information for debugging.

1. The RUN block - This contains all of the information required to run the program. The operating system will use this part of the file for segment one of the running program. This area contains the code sections and the initial value records used to initialize the static areas when the program is started.

2. The SYMBOLIC block - This contains information to aid in run time debugging of the program. This area is used by the debug facilities in the HELP processor.

3. The LINKAGE block - This information is used by the linker program. After adding or deleting sections, the linker program will relocate the address constants in the program using this information.

LINKER PROCESSING

When the linker program processes an object program, it will perform the following operations to the basic parts of the object program:

1. The RUN block - It can add new sections to either the code or the static areas. These must be added as a whole section. It can also delete whole sections or reorder the sections, again only as whole sections, never as a part of a section. Locations within the sections will not be inspected or changed other than relocatable address constants having their values adjusted by the linker. In some circumstances, the linker may have to change a flag in one of the relocation records. When a section is deleted from a program, all relocation records that reference that section must have the flag turned off. If a static section is added, all references to it must be examined. If the reference is in a code section, it must be an R type (relocation address constant) of address constant. If not, it will be flagged as an error.

   Only the terms add or delete have been used. Although the linker can be used to replace a section, this should be considered a deletion of the current section and an addition of the new section.

2. The LINKAGE block - When a code or static section is added or deleted, the linker will add or delete the corresponding linkage section. It will also adjust the object time and run time starting address of the sections that follow. When adding a section, all references to it are marked as resolved and relocated. If a section is deleted, all references are marked as unresolved and are relocated relative to hexadecimal F00000.

3. The SYMBOLIC block - If a section is added or deleted, the linker will add or delete the corresponding symbolic section. It will not inspect or change any of the records within a section for any reason.

RUN PROCESSING

A program is invoked by the command processor or by another program using the LINK facility of the operating system. First, the run portion of the program is made addressable as segment 1. (The first location will be location 1,048,576.) The system will then double word align the stack, find out the run time length of the static area and push this much space onto the stack. The start of this area will be passed to the program in register 14. The initial value records will then be processed. When an origin record is encountered, the origin displacement will be added to the value in register 14 and this will be used as the starting location of the section. When either text or relocation records are read, their displacement will be added to this value and moved to the location calculated. If a relocation record has the relocate flag on, the value in register 14 is to be added to the initial value in the record.

APPENDIX C DATA MANAGEMENT SYSTEM MESSAGES

Appendix C contains the following types of messages, listed in the order they appear in the appendix:

1. SVC OPEN Cancel Messages.
2. SVC OPEN Respecify Messages.
3. DMS Function-Request Cancel Messages.
4. SVC CLOSE Cancel Messages.
5. File Status (FS) Codes for DMS and ADMS.

The following types of messages are not included in this appendix:

1. Messages issued by program 'BUILDALT' for OUTPUT mode creation of alternate indexed files (acknowledge and cancel messages).
2. Miscellaneous acknowledge messages from SVC OPEN and DMS function requests.

VS DMS No-Message Option

The No-Message Option is available in SVC OPEN, SVC CLOSE, and DMS. This option causes the suppression of messages normally appearing on the workstation screen.

If UFBF4NOMSG = 1, the file status for the operation is set equal to C'60'. For SVC OPEN and SVC CLOSE, the message ID is stored in the first four bytes of the UFB. Return is made using the address in UFBERRAD; if this address is zero, UFBF4NOMSG is ignored.

1. SVC OPEN Cancel Messages

These messages deal primarily with invalid information supplied in the UFB. Some also refer to unusual conditions that rarely arise during normal SVC OPEN usage; for example, UPDATFDR SVC errors, I/O errors when reading AXD1 blocks, etc.

NOTE

There is no continuation possible when these messages are issued.

ERROR NUMBER	MESSAGE
E000	INVALID UFB ADDRESS PRESENTED TO SVCOPEN.
E001	DEVICE CLASS (XX) = INVALID OPEN MODE (XX) = INVALID FILE ORGANIZATION (XX)= INVALID RECORD SIZE = INVALID RECORDS ARE FIXED LENGTH. KEY SIZE = XXX KEY POSITION = INVALID
E002	FILE ALREADY OPEN (UFBF1OPEN SET)
E006	TASK WORKSTATION NOT AVAILABLE.
E007	MAXIMUM NUMBER OF FILES ALREADY OPEN
E011	REQUIRED BUFFER(S) NOT AVAILABLE FOR FILE PROCESSING
E014	UNEXPECTED DEALLOCATION ERROR FOR MAGTAPE DEVICE.
E016	BACKGROUND TASK ATTEMPTED TO OPEN WORKSTATION. THE WORKSTATION MAY BE OPENED IN FOREGROUND ONLY.
E017	INVALID OPEN MODE FOR PHYSICAL ACCESS METHOD. (EXTEND AND SHARED MODES ARE INVALID.)
E018	THE PROGRAM IS REQUESTING AN INVALID OPEN MODE (SHARED, EXTEND) FOR A FILE RESIDING ON AN UNSTRUCTURED DISKETTE VOLUME.
E019	THE PROGRAM IS REQUESTING AN INVALID MODE (EXTEND MODE) FOR INDEXED FILE PROCESSING.
E021	THE PROGRAM IS REQUESTING AN INVALID FILE ORGANIZATION (INDEXED) FOR A FILE RESIDING ON AN UNSTRUCTURED DISK VOLUME.
E023	INVALID ACCESS METHOD SPECIFICATION IN UFB (UFBF1).
E024	BLOCK ALLOCATION ERROR. SPACE NOT AVAILABLE ON VOLUME AND EXIT-OPTION NOT IN USE.
E025	THE PROGRAM IS REQUESTING AN INVALID MODE (SHARED MODE) FOR FILE PROCESSING UNDER THE BLOCK ACCESS METHOD (BAM).
E027	SHARING TASK NOT ACTIVE.
E028	UNABLE TO GET UNIQUE PORT NAME.
E029	SHARER RESPONSE CODE = XX-YYYY. UNEXPECTED ERROR HAS OCCURRED WHILE OPENING A FILE FOR SHARED ACCESS.
E030	INVALID BUFFER POOL SPECIFICATION. (ACCESS METHOD SUPPLIED IS INVALID.) BUFFER POOLING CAN ONLY BE USED WITH INDEXED FILES IN INPUT OR IO MODE.
E031	THE BUFFER POOL TABLE ADDRESS SUPPLIED (IN UFBBUFSTART) IS INVALID.
E032	THE BUFFER POOL TABLE HAS NOT BEEN CORRECTLY INITIALIZED.
E033	THE BUFFER COUNT SUPPLIED BY THE PROGRAM IS TOO SMALL. THE MINIMUM BUFFER COUNT IS 3.
E034	AN UNEXPECTED ERROR HAS OCCURRED WHIE UPDATING THE FILE LABEL. THE FILE CANNOT BE SUCCESSFULLY OPENED FOR UPDATE.
E035	THE ALTERNATE INDEX BLOCK (AXD1) ADDRESS SUPPLIED IS INVALID.

ERROR NUMBER	MESSAGE
E036	THE ALTERNATE INDEX COUNT IN UFBALTCNT IS INCORRECT.
E037	ALTERNATE INDEX INFORMATION FOR FILE CREATION IS INCORRECT.
E038	UNABLE TO READ AXD1 BLOCK FROM FILEBLOCK 0.
E039	FAIL TO FREE THE BUFFER AFTER READING AXD1 FROM FILE BLOCK 0.
E040	ALTERNATE INDEX KEYSIZE PLUS PRIMARY KEYSIZE TOO LARGE.
E050	UNABLE TO ALLOCATE SYSTEM MEMORY--GETMEM FAILURE.
E055	UNABLE TO SET UP THE MAPPING CONTROL TABLE (MCC) FOR THE ADMS FILE IN SHARED MODE.
E056	THE ADDRESS OR SIZE OF THE MAPPING CONTROL TABLE (MCC) FOR THIS ADMS FILE IS INVALID.
E057	UNABLE TO EXTRACT INFORMATION FROM DATA DICTIONARY TO SET UP THE UFB FOR THIS ADMS FILE.
E058	THE PROGRAM HAS ATTEMPTED TO PROCESS AN ADMS FILE, BUT THE VIEW OF THE FILE IS NOT OPENED.
E059	THE VIEW OF THIS FILE DOES NOT HAVE THE EXLCUSIVE ACCESS RIGHT TO OPEN IT IN NON-SHARED MODE.
E060	THE PROGRAM HAS ATTEMPTED TO OPEN AN ADMS FILE, BUT THE FILE SPECIFIED IS NOT AN ADMS FILE.
E061	ADMS DATA DICTIONARY ERROR--INCORRECT FILE ATTRIBUTE FOUND.
E062	THE PROGRAM HAS ATTEMPTED TO OPEN AN EXISTING ADMS FILE IN A DATABASE WITH RECOVERY. BUT THE DATABASE HAS NOT BEEN BACKED UP, AND RECOVERY CANNOT WORK WITHOUT THE BACKUP. PLEASE RUN DATABASE BACKUP.
E063	THE PROGRAM HAS ATTEMPTED TO CREATE AN ADMS FILE IN A DATABASE WITH RECOVERY BUT OUTPUT MODE WITH RECOVERY IS NOT ALLOWED AFTER SCHEMA BACKUP. THE SCHEMA MUST BE RE-ACTIVATED W/O RECOVERY.
E064	ADMS FILE CREATION ERROR--UNABLE TO RECORD THE CREATION OF THIS FILE IN THE DATA DICTIONARY.
E065	THE ADMS FILE TO BE OPENED IS NOT YET LOADED INTO THE DATABASE.
E082*	UNABLE TO COMMUNICATE WITH SHARER TASK TO OBTAIN SCHEMA STATUS.
E083*	THIS FILE IS PART OF AN ADMS DATABASE WHICH HAS ENCOUNTERED A CRASH CONDITION. RESTORE MUST BE RUN BEFORE THE FILE CAN BE SUCCESSFULLY OPENED IN ADMS MODE.

* This is a new error message added to OS Release 5.1. Older OS releases do not generate this error.

2. SVC OPEN Respecify Messages

These messages deal with situations where the user may successfully continue either by supplying additional information or by correcting information already supplied. Situations involving possession conflicts or volume mounting are also handled by these respecification messages. The user may always continue after a respecify message.

ERROR NUMBER	MESSAGE
R001	FILE IDENTIFICATION INFORMATION IS INCOMPLETE. PLEASE SUPPLY THE MISSING INFORMATION BELOW.
R002	PLEASE SUPPLY THE APPROXIMATE NUMBER OF RECORDS IN THE DISK FILE TO BE CREATED. THIS VALUE WILL BE USED FOR INITIAL DISK-SPACE ALLOCATION.
R003	DEVICE SPECIFIED IS UNKNOWN OR NOT SUPPORTED. PLEASE RESPECIFY.
R004	DEVICE SPECIFIED IS INVALID FOR THIS PROCESSING MODE. PLEASE RESPECIFY.
R005	THE PROGRAM IS NOT REQUESTING A CONSECUTIVE-PRINT FILE. THEREFORE, THE FILE CANNOT BE ASSIGNED TO A PRINTER. PLEASE SPECIFY ANOTHER DEVICE TYPE.
R006	DEVICE NUMBER INCORRECTLY SPECIFIED. PLEASE RESPECIFY DEVICE (E.G. PRINTER 3). (NOTE--DEVICE NUMBER IS OPTIONAL AND MAY BE OMITTED).
R007	DEVICE NUMBER DOES NOT CORRESPOND TO DEVICE CLASS. PLEASE RESPECIFY DEVICE.
R008	NO PRINTER CURRENTLY AVAILABLE. ASSIGN OUTPUT TO DISK OR FREE PRINTER FOR ALLOCATION.
R013	THE FILE BELOW IS ALREADY OPENED BY THIS PROGRAM. PLEASE SPECIFY ANOTHER FILE.
R014	UNEXPECTED READFDR SVC ERROR.
R014	FILE SPECIFIED NOT FOUND IN LIBRARY. PLEASE RESPECIFY FILENAME.
R014	LIBRARY NOT FOUND IN VOLUME TABLE OF CONTENTS. PLEASE RESPECIFY LIBRARY.
R016	THE FILE SPECIFIED IS IN USE AS A SYSTEM-ONLY PAGING FILE. PLEASE RESPECIFY.
R018	THIS FILE IS CURRENTLY IN USE AS A PROGRAM FILE. THEREFORE, IT CAN ONLY BE OPENED IN INPUT MODE. PLEASE RESPECIFY THE FILE.
R020	UNEXPECTED CREATFDR SVC ERROR.
R020	FILE SPECIFIED ALREADY EXISTS. PLEASE RESPECIFY FILE.
R020	VTOC FULL, NO ROOM FOR FILE LABEL. PLEASE SPECIFY ANOTHER VOLUME.
R020	VOLUME FULL, NO ROOM FOR FILE. PLEASE SPECIFY ANOTHER VOLUME OR USE A SMALLER FILE SIZE.
R021	INVALID INFORMATION IN FILE LABEL. PLEASE RESPECIFY FILE.
R022	THE TAPE SPECIFIED BELOW IS AN NL-TAPE, BUT PROGRAM REQUIRES A TAPE WITH A DIFFERENT LABEL TYPE. PLEASE RESPECIFY.
R024	THE FILE AT POSITION XXX WITHIN THE TAPE VOLUME IS XXXXXXXXXXXXXX. THIS DOES NOT AGREE WITH THE FILE SPECIFIED BELOW. PLEASE RESPECIFY.
R025	THE DEVICE SPECIFIED IS ALREADY IN USE BY THIS PROGRAM. PLEASE RESPECIFY.
R026	THE DEVICE SPECIFIED HAS BEEN LOGICALLY DETACHED AND IS THEREFORE NOT AVAILABLE. PLEASE RESPECIFY.
R027	THE PROGRAM REQUIRES XXXXXXXXXX. THE FILE SPECIFIED BELOW IS XXXXX XXXXX. PLEASE RESPECIFY.

ERROR NUMBER	MESSAGE
R028	THE PROGRAM REQUIRES A FILE CONTAINING XXXXX-CHARACTER RECORDS. THE FILE SPECIFIED BELOW CONTAINS XXXXX-CHARACTER RECORDS. PLEASE RESPECIFY.
R029	A FILE SEQUENCE NUMBER OF ZERO IS INVALID. PLEASE RESPECIFY.
R030	TAPE IO ERROR OCCURRED DURING TAPE POSITIONING OR LABEL PROCESSING. IOSW = XXXXXXXX XXXXXXXX. PLEASE RE-MOUNT THE TAPE VOLUME IN ORDER TO TRY AGAIN.
R031	THE TAPE VOLUME IS WRITE-PROTECTED, AND THEREFORE CANNOT BE PROCESSED IN OUTPUT OR EXTEND MODE. PLEASE PUT A WRITE-ENABLE RING ON THE TAPE, AND RE-MOUNT THE VOLUME, OR USE (ENTER) TO RESPECIFY.
R032	THE UNSTRUCTURED DISKETTE VOLUME SPECIFIED FOR OUTPUT IS CURRENTLY IN USE. PLEASE RESPECIFY.
R033	THE PROGRAM IS REQUESTING A FILE THAT RESIDES ON AN UNSTRUCTURED DISK VOLUME. THE FILE SPECIFIED BELOW RESIDES ON A DISK VOLUME WITH A VTOC. PLEASE RESPECIFY.
R034	THE INDEXED FILE SPECIFIED BELOW CAN NOT BE PROCESSED IN EXTEND MODE. PLEASE RESPECIFY. (EXTEND MODE IS ONLY SUPPORTED FOR CONSECUTIVE FILES.)
R035	THE INDEXED FILE SPECIFIED BELOW WAS NOT CLOSED AT FILE CREATION. THE FILE IS CURRENTLY NOT USEABLE AND SHOULD BE RE-CREATED. PLEASE SPECIFY ANOTHER FILE.
R036	THE FILE SPECIFIED BELOW WAS NOT CLOSED AT FILE CREATION. THEREFORE, THE FILE LABEL INDICATES THAT THE FILE CONTAINS NO RECORDS. IF YOU WISH TO ACCESS THE WHOLE FILE SPACE (MAXIMUM NUMBER OF RECORDS), USE PF2 AND THE END-OF-FILE INDICATOR WILL BE SET ACCORDINGLY. OTHERWISE, PLEASE SPECIFY ANOTHER FILE.
R037	CODE = XX; UNEXPECTED OUTPUT-FILE SCRATCH ERROR. PLEASE SPECIFY ANOTHER OUTPUT FILE NAME IN ORDER TO CONTINUE.
R038	UNABLE TO FIND FILE SPACE ON ANY ELIGIBLE VOLUME. PLEASE SPECIFY A SMALLER FILE, USE A PRIVATE VOLUME, OR RELEASE (THROUGH SCRATCH) THE REQUIRED DISK SPACE.
R039	THE DISKETTE VOLUME SPECIFIED BELOW IS WRITE-PROTECTED. PLEASE RE-MOUNT THIS DISKETTE WITH WRITE-ENABLED, OR SPECIFY ANOTHER FILE.
R040	THE FILE SPECIFIED BELOW ALREADY EXISTS. USE PF3 IF YOU WISH TO SCRATCH THE EXISTING FILE AND CONTINUE. OTHERWISE, PLEASE SPECIFY ANOTHER FILE NAME.
R041	THE FILE SPECIFIED BELOW IS CURRENTLY IN USE AND CANNOT BE SCRATCHED. PLEASE SPECIFY ANOTHER OUTPUT FILE NAME.
R045	ENTER KEY USED WITH INVALID DEVICE SPECIFICATION BELOW. USE PF4 KEY FOR MOUNT OPERATION. IF A MOUNT OPERATION IS NOT REQUIRED, PLEASE USE THE ENTER KEY WITH DEVICE = DISK.
R047	SHARER RESPONSE CODE = XX-YYYY. CONSULT SHARER ERROR LIST FOR EXPLANATION. PLEASE SPECIFY ANOTHER FILE IN ORDER TO CONTINUE.
R048	INVALID VALUE ENTERED FOR PRINTER OPTION. FORM # MUST BE LESS THAN 256. PRTCLASS MUST BE A LETTER (A-Z). COPIES MUST BE A NUMBER BETWEEN 1 AND 32,767. PLEASE RESPECIFY.
R049	THE FILE SPECIFIED BELOW IS A PROGRAM FILE WITH SPECIAL ACCESS RIGHTS. ONLY A SECURITY ADMINISTRATOR MAY MODIFY THIS FILE. PLEASE RESPECIFY.
R049	THE CURRENT USER DOES NOT HAVE THE REQUIRED ACCESS RIGHTS FOR THE FILE SPECIFIED BELOW. PLEASE RESPECIFY.

ERROR NUMBER	MESSAGE
R050*	THIS FILE IS A PARTIAL FILE CREATED BY BACKUP, AND MAY BE OPENED IN BAM OR PAM, WITH THE PARTIAL FILE FLAG SET. PLEASE RESPECIFY.
R051	THE SHARER HAS RUN OUT OF MEMORY FOR ITS CONTROL BLOCKS. THIS FILE MAY BE OPENED SUCCESSFULLY AFTER ENOUGH MEMORY HAS BEEN RELEASED (BY OTHER SHARED USERS).
R052	THE FILE BELOW IS ALREADY OPENED IN SHARED MODE BY THIS PROGRAM. PLEASE SPECIFY ANOTHER FILE.
R053	FILE SPECIFIED NOT FOUND IN LIBRARY. PLEASE RESPECIFY FILENAME.
R054	LIBRARY NOT FOUND IN VOLUME TABLE OF CONTENTS. PLEASE RESPECIFY LIBRARY.
R059	THE VOLUME SPECIFIED IS MOUNTED FOR EXCLUSIVE USE. A FILE ON AN EXCLUSIVE VOLUME MAY NOT BE SHARED. PLEASE SPECIFY ANOTHER FILE (OR RE-MOUNT THIS VOLUME).
R060	THE PROGRAM REQUIRES A FILE WITH A DIFFERENT FILE-ORGANIZATION FROM THE FILE SPECIFIED BELOW. PLEASE RESPECIFY.
R061	THE PROGRAM REQUIRES A FILE WITH A DIFFERENT RECORD SIZE FROM THE FILE SPECIFIED BELOW. PLEASE RESPECIFY.
R062	THE DISK VOLUME SPECIFIED IS NOT MOUNTED. PLEASE MOUNT THE DISK VOLUME OR RESPECIFY.
R063	THE FILE SPECIFIED BELOW IS CURRENTLY IN NON-SHARED USE. PLEASE RESOLVE THIS POSSESSION CONFLICT OR RESPECIFY.
R064	THE CURRENT USER DOES NOT HAVE THE REQUIRED ACCESS RIGHTS TO SCRATCH THE FILE SPECIFIED BELOW. PLEASE SPECIFY ANOTHER FILE.
R065	THE RETENTION PERIOD FOR THE FILE SPECIFIED BELOW HAS NOT EXPIRED. THE FILE CANNOT BE SCRATCHED UNLESS THE EXPIRATION DATE IS MODIFIED. PLEASE SPECIFY ANOTHER FILE OR USE THE COMMAND PROCESSOR TO MODIFY THE EXPIRATION DATE AND SCRATCH THIS FILE.
R066	THE CONSECUTIVE FILE SPECIFIED BELOW CAN NOT BE OPENED IN SHARED MODE. PLEASE RESPECIFY.
R067	THE FIRST CHARACTER OF A LOG-FILE BEING OPENED IN SHARED MODE MAY NOT BE "#". PLEASE RESPECIFY
R068	THE PROGRAM WILL NOT ACCEPT THIS FILE FROM TAPE. PLEASE RESPECIFY.
R069	END OF TAPE REACHED WHILE POSITIONING TAPE BY FILE SEQUENCE NUMBER.
R070	VOLUME FULL, UNABLE TO ADD ANOTHER FILE ON THE TAPE. PLEASE RESPECIFY.
R071	THE TAPE FILE SPECIFIED BELOW IS NOT ON THE TAPE VOLUME. PLEASE RESPECIFY.
R072	THE DEVICE SPECIFIED IS NOT A TELECOMMUNICATION DEVICE. PLEASE RESPECIFY.
R073	CONTROL BLOCKS (PPB, LCB) FOR THIS TC DEVICE ARE NOT PROPERLY SET UP. PLEASE RESPECIFY.
R074	UNABLE TO LOAD THE MICROCODE FOR THIS TC DEVICE. PLEASE RESPECIFY.
R075	UNABLE TO CONNECT THE TC LINE, OR INCORRECT CONNECT PARAMETERS SUPPLIED. PLEASE RESPECIFY.
R076	THE PROGRAM HAS SUPPLIED AN INVALID ADDRESS FOR THE CONNECT PARAMETER. PLEASE RESPECIFY.
R077	THE TAPE VOLUME IS NOT THE CORRECT SEQUENTIAL VOLUME FOR THIS TAPE FILE. PLEASE RESPECIFY.
R078	EXTEND MODE PROCESSING FOR IBM LABELED TAPE IS NOT SUPPORTED. PLEASE RESPECIFY.

ERROR NUMBER	MESSAGE
R080	THE PROGRAM HAS ATTEMPTED TO OPEN A RE-RESTART FILE, BUT THE FILE SPECIFIED IS NOT A RESTART FILE. PLEASE RESPECIFY.
R081	THE PROGRAM HAS ATTEMPTED TO MODIFY THE ADMS FILE NAME AND ITS ATTRIBUTES. BUT THESE ATTRIBUTES ARE ASSIGNED BY THE DATA DICTIONARY AND CANNOT BE CHANGED AT OPEN. THEIR CORRECT VALUES ARE DISPLAYED BELOW, PRESS (ENTER) TO CONTINUE.

* With the introduction of OS Release 5.1, the text for error number R050 is modified as follows:

THIS FILE IS A PARTIAL FILE CREATED BY BACKUP FOR USE BY RESTORE. IT MAY BE OPENED ONLY IN BAM OR PAM, WITH THE PARTIAL FILE FLAG SET. PLEASE RESPECIFY.

3. DMS Function-Request Cancel Messages

The file status message (ID = 000) covers all file status values including cases where the significance of the FS value is determined by additional factors such as current function request and file organization. The file status message appears as a cancel message if UFBERRAD = 0. Otherwise, an acknowledge message is issued before taking the error exit. (The acknowledge message may be masked out by using UFB4NOACK.)

Other DMS cancel messages reflect unusual conditions caused mainly by incorrect user modification of UFB fields, unexpected errors, or invalid block contents for indexed file processing.

These messages may be issued as a result of any one of the five DMS function requests or by the DMS CLOSE vector (for the last IO operation on the file).

ERROR NUMBER	MESSAGE	POSSIBLE CAUSE
000	ERROR DETECTED AND USER ERROR EXIT NOT IN USE. FILE STATUS = XX.	Check meaning of File Status code for cause of error message. (Refer to page 354.3.)
001	INVALID FIELD FOUND WHILE PROCESSING FILE X (UFBBCBFLAGS).	Invalid buffer status flags in the UFB.
002	INVALID BLOCK NUMBER DETECTED BY SVC XIO (UFBBUFBLOCK) WHEN ATTEMPTING DISK I/O.	UFBBUFBLOCK contains invalid data. Can be caused by invalid data in Data Link Chain of the prior block.
003	INVALID FIELD FOUND WHILE PROCESSING FILE X (UFBRECSIZE=0).	Cannot have files with record lengths of zero.
004	INVALID BUFFER POOL INFORMATION DETECTED AT BEGINNING OF FUNCTION REQUEST.	There is an error in the Buffer Control Entry.
005	INVALID OFB POINTER FOUND (UFBOFB).	OFB Pointer in the UFB contains an address which is not an OFB.
006	UNEXPECTED ERROR FOUND WHILE ATTEMPTING TO ALLOCATE ADDITIONAL DISK SPACE.	SVC return code is incorrect. This is an indication of a serious DMS problem.
007	VTOC I/O ERROR OCCURED DURING SVC ALEX.	A VTOC IO error occurred while trying to obtain an additional extent. ALEX is an acronym for Allocate Extent.
008	UNABLE TO ALLOCATE DISK EXTENT SINCE ALL BUFFERS OR GETMEM POOL IN USE.	ALEX return code = 20. No work space available for UPDATFDR.
009	MAG TAPE READ OPERATION FAILED; NO DATA WAS TRANSFERRED.	Residual count greater than or equal to block size--probable IOP firmware error.
010	FUNCTION-REQUEST ISSUED ON NON-OPENED FILE.	Before performing a task within a file, the file must first be opened.
011	SECOND PHYSICAL I/O OPERATION ISSUED ON FILE WITHOUT WAITING FOR PREVIOUS I/O COMPLETION.	Occurs when two XIO's in a row were performed with no CHECK operation between them.
012	UNUSED	
013	ERROR FOUND WHILE READING FILE INDEX. THIS FILE SHOULD BE REORGANIZED IN ORDER TO GENERATE THE INDEX CORRECTLY.	Invalid condition exists in the index block currently being read.

ERROR NUMBER	MESSAGE	POSSIBLE CAUSE
014	INVALID RECORD FORMAT DETECTED. ERROR OCCURRED WHEN EXPANDING COMPRESSED DATA RECORD.	A record within a block contains garbage.
015	INVALID BLOCK NUMBER FOUND WHILE BUILDING OR UPDATING THE FILE INDEX. FILE INDEX.	Occurs when contents of a Data Link Chain in a data block contains incorrect data. This error generated when data length in block exceeds 7FC.
016	RESIDUAL COUNT NOT ZERO AFTER REWRITE OPERATION (LARGE BUFFER).	On a rewrite all bytes are subtracted from the block length, the difference must equal zero. This error occurs when it is not.
017	INVALID UFB FIELD FOUND FOR REWRITE OPERATION (OFFSET=0).	User damaged segment 2 (UFB) data.
018	BUFFER POOL ERROR DETECTED. LOCKED BUFFER (BCE) IN CONTROL TABLE DOES NOT AGREE WITH CURRENT BUFFER (BCB).	User damaged segment 2 (UFB or BCT) data.
019	BLOCK TYPE (BCE) IN BUFFER POOL DOES NOT AGREE WITH CURRENT READ REQUEST.	Buffer Control Entry in the buffer pool is invalid.
020	BUFFER POOL ERROR DETECTED. BLOCK TYPE (BCE) IS INVALID FOR IO INITIATION.	Block Type in the buffer pool is invalid.
021	BUFFER POOL ERROR DETECTED. BUFFER (BCE) WITH IO IN PROGRESS NOT ON BCTBL CHAIN OR INTERNAL LOCK OTHER BCE OPERATION FAILED.	User damaged UFB or BCT.
022	ERROR DETECTED IN THE ALTERNATE INDEX DATA STRUCTURE, UNABLE TO LOCATE THE RECORD.	Alternate index block contains a primary key value which is not in any data block.
023	TRACE ROUTINE FOR DUPLICATE KEY VALUES FAILED. ALTERNATE TREE NOT MODIFIED.	The offset into the alternate index block is invalid for a user.
024	UNEXPECTED ERROR OCCURED DURING FILE RESTORE OPERATION. WRITE OR REWRITE FUNCTION FOR ALTERNATE INDEX FILE FAILED DUE TO DUPLICATE KEY ERROR, AND ATTEMPT TO RESTORE FILE WAS UNSUCCESSFUL.	On a WRITE or REWRITE to an alternate-indexed file, if a duplicate key is encountered on a path with no duplicates allowed, the system attempts to delete the record from the primary tree and all alternate trees on which it has been written. This error occurs if the attempt fails unexpectedly. User should attempt COPY/REORG.

4. SVC CLOSE Cancel Messages

These messages refer to unexpected error conditions that rarely occur.

ERROR NUMBER	MESSAGE
E001	SVC CLOSE ISSUED FOR NON-OPENED FILE.
E002	DEALLOCATION ERR; OFB NOT FOUND.
E003	DEALLOCATION ERR; IORE QUEUED.
E004	CODE = ; UPDATFDR SVC ERR. NO DEALLOCATION.
E005	UNABLE TO DEALLOCATE BUFFER DUE TO INVALID BUFFER ADDRESS OR BUFFER LENGTH IN UFB.
E006	INVALID UFB POINTER RETURNED AFTER LAST DMS OPERATION.
E007	CODE = XX; SCRATCH SVC ERROR. FILE CLOSED OK, BUT NOT SCRATCHED.
E008	INVALID UFB ADDRESS PRESENTED TO SVC CLOSE.
E009	UNEXPECTED ERROR OCCURRED WHILE ATTEMPTING TO CLOSE FILE (SHARED MODE).
E010	UNABLE TO DEALLOCATE BUFFER WITHIN BUFFER POOL DUE TO INVALID ADDRESS OR LENGTH IN BUFFER CONTROL TABLE ENTRY.
E012	FAIL TO LOCATE PROGRAM BUILDALT. UNABLE TO BUILD ALTERNATE INDEXES.
E013	FILE LABEL NOT UPDATED ( ). USER PROGRAM HAS INCORRECTLY MODIFIED THE UFB.
E014	FILE LABEL NOT UPDATED ( ). VTOC ERROR DETECTED.
E015	THE PROGRAM TRIED TO CLOSE THE ADMS FILE WHILE IN THE TRANSACTION STATE.

5. FILE STATUS (FS) CODES FOR DMS AND ADMS

DMS and ADMS return to the user program by means of the RETURN macroinstruction. User registers 2 through 15 are always restored. Register 0 (R0) is also restored unless UFBEODAD or UFBERRAD is used--R0 then contains the normal return address. Register 1 (R1) is also restored unless the Read-No-Data option has been used--R1 then contains the record address.

DMS and ADMS indicate the result of the function request through file status bytes UFBS1 and UFBS2. These bytes generally contain a value of X'30' - X'39', corresponding to the ASCII characters 0 through 9, called the File Status (FS) Code. File Status Byte 1 (UFBS1) indicates the general type of file status and File Status Byte 2 (UFBS2) indicates a specific item within the group. The various groups are defined as follows:

0 - Successful Completion.
1 - End of File.
2 - Record Not Found (Disk File).
3 - IO Error or Boundary Violation.
4 - ADMS Codes.
6 - Cancel.
7 - Time-Out.
8 - Special Shared Mode Errors.
9 - Miscellaneous - This Group includes errors caused by incorrect user-supplied information; e.g., Invalid Function, Invalid Mask, Invalid Length, or Invalid Format.

Following is a list of File Status codes with a description of each code and the conditions under which it can occur.

FILE STATUS FOR NORMAL RETURNS

FS CODE	MEANING	FUNCTION REQUEST	DEVICE	FILE ORGANIZATION	MODE	CAUSE
00	Successful Completion.	N/A	Disk, Tape, Printer	N/A	N/A	N/A
0X	Successful Completion.	N/A	Workstation	N/A	N/A	UFBFS2 ('X' in code field) contains the AID byte.
02	Successful Completion.	Read	Disk	Alternate Indexed	N/A	After successfully completing a READ KEYED or READ NEXT on an alternate key path, the return code is 02 indicating at least one more record exists with the same alternate key value.

FILE STATUS FOR UFBEODADRETURN

FS CODE	MEANING	FUNCTION REQUEST	DEVICE	FILE ORGANIZATION	MODE	CAUSE
10	End of File Reached.	READ NEXT	Disk or Tape	N/A	Input, I/O, or Shared	End of file was reached.

FILE STATUS FOR UFBEODADRETURN (cont'd)

FS CODE	MEANING	FUNCTION REQUEST	DEVICE	FILE ORGANIZATION	MODE	CAUSE
11	End of Volume.	READ NEXT	Tape	N/A	N/A	This code is returned if the user program indicates (by UFBTFLGEODEOV) that no automatic volume switch is desired.
21	Record Key Out of Sequence or Duplicate Key Found During Indexed File Creation.	WRITE	Disk	Indexed	Output	The current record key is not greater than the preceding record key.
22	Duplicate Key Value.	WRITE	Disk	Indexed	I/O or Shared	The record to be added to the file has the same key as an existing record in the file.
23	Record Not Found in File.	READ RELATIVE	Disk	Consecutive	Input or I/O	The supplied record number is equal to zero or greater than the highest record number in the file.
23	Record Not Found in File.	READ KEY or START (Equal)	Disk	Indexed	Input, I/O*, or Shared	There is no record in the file containing a key equal to the supplied key.
23	Record Fot Found in File.	READ or REWRITE	Disk	N/A	I/O	The supplied block number is beyond the end of the file.
24	Primary Extent Exceeded (Indexed File Creation).	WRITE	Disk	Indexed	Output	Primary extent exceeded. The record cannot be added to the file. The file may be closed successfully and then opened in I/O Mode to add more records.
24	Record Not Found. Key Supplied Greater Than Key Value in File.	START (Greater Than) or (Greater Than or Equal)	Disk	Indexed	Input, I/O, or Shared	The supplied key is greater than the highest key value in the file.

FILE STATUS FOR UFBERRAD RETURN

FS CODE	MEANING	FUNCTION REQUEST	DEVICE	FILE ORGANIZATION	MODE	CAUSE
30	Permanent IO Error IOSW = XXXXXXXX XXXXXXXX.	N/A	N/A	N/A	N/A	A physical I/O operation was attempted and a hardware error occurred. The error is logged separately by SVC CHECK. This file status is returned for hardware errors only; it is not returned for program related errors.
34	Workstation Order Check.	READ or REWRITE	Workstation	N/A	I/O	Invalid information supplied in the workstation order area; i.e., Invalid Cursor Position: Row 25 Column 10.
34	Boundary Violation (Extent Cannot Be Obtained).	WRITE	Disk	Consecutive	Output or Extend	There is no more space in the file for additional records. An additional extent is unavailable because either the maximum number of extents are already allocated or the extent size is not available on volume.
34	Boundary Violation (Extent Limit of 13 Has Been Reached).	WRITE	Disk	Indexed	I/O or Shared	There is no more space in the file for additional records (as above) due to extent limit (13) exceeded or no available extent on volume. For Shared mode, an additional extent may also be unavailable due to maximum number of additional extents per run already allocated.
60	DMS Cancel Condition Occurred; Cancel Message Suppressed.	N/A	Disk or Tape	N/A	Shared	User requested suppression of all DMS-Cancel messages. Process the file in non-Shared mode to set the error message flag. If a DMS error condition code with FS=60-019 occurs, refer to DMS error code 019 in Appendix C.
70	Shared Time-Out Condition.	N/A	N/A	N/A	N/A	This feature will be available with the Advanced Sharer.

FILE STATUS FOR UFBERRAD RETURN (cont'd)

FS CODE	MEANING	FUNCTION REQUEST	DEVICE	FILE ORGANIZATION	MODE	CAUSE
80	Invalid Key Area Found for Read Key or Start Key.	READ KEYED or START KEYED	N/A	Indexed	Shared	UFBKEYAREA does not point to the key embedded in the record; i.e., specifies the key has a value of one for a length of five but it actually has a value of two for a length of five.
81	Invalid Read No-Data Issued.	READ NO-DATA	Disk	Indexed or Consecutive	Shared	Attempting to do a Read No-Data in Shared mode which is an invalid function request.
82	Label Update Operation after Last Function Was Unsuccessful.	N/A	N/A	N/A	Shared	Internal error by DMS. The file label (FDR1) is updated whenever any of the following fields are modified by DMS: Root Block Number; First Data Block Number; or Count of Levels in the (Primary) Index. If UPDATFDR is unsuccessful, FS equals 82 is returned.
83	The Sharing Task Has Terminated and Must be Restarted.	N/A	N/A	N/A	Shared	Sharing task is functioning incorrectly. Must IPL the System to restart Sharer.
84	Invalid Record Size or Area Supplied for Shared Request.	N/A	N/A	N/A	Shared	User Attempted to rewrite a variable length record whose length is greater than the maximum record size specified in the VTOC.
85	Update Access Denied.	WRITE, REWRITE, or DELETE	N/A	N/A	Shared	User Attempted to update a file in Shared mode but has Read-Only access.
86	Resource Control Error.	N/A	N/A	N/A	Shared	Incorrect sequence of Shared function requests; e.g., Attempting to do a Start Hold on a file while another file is already held.
95	Invalid Function Sequence.	REWRITE, DELETE, or READ NEXT HOLD	Disk	Indexed	IO or Shared	Invalid function sequence similar to consecutive file case above occurred. Also returned if Read Next Hold issued without a file block HELD (invalid sequence).
95	READ RELATIVE Invalid for Variable Length Records.	READ RELATIVE	Disk	Consecutive	Input, or IO	Read Relative is only valid for fixed-length consecutive files.

FILE STATUS FOR UFBERRAD RETURN (cont'd)

FS CODE	MEANING	FUNCTION REQUEST	DEVICE	FILE ORGANIZATION	MODE	CAUSE
95	Invalid Function Request.	N/A	N/A	N/A	N/A	Valid function requests are described for the given combinations of device class, open mode and file organization supported by DMS. After a file has been opened, an invalid function request is flagged with FS equals 95. Example: attempting to write a record while the file is opened in Input mode.
95	Invalid Function Sequence.	REWRITE	Disk	N/A	Shared	Record was not read with the HOLD option. For Shared Mode, an intervening READ with HOLD on another file may have released the HOLD. A function sequence error exists since the record cannot be rewritten unless it is 'HELD'.
95	REWRITE Function Invalid for Consecutive File with Compressed Records.	REWRITE	Disk	Consecutive	IO	Consecutive files can be rewritten only for fixed-length records.
95	Invalid Function Issued on Alternate Indexed File.	N/A	N/A	N/A	N/A	N/A
95	READ NEXT Issued on Indexed File when Current Position Was Undefined.	N/A	N/A	N/A	N/A	N/A
95	Invalid Function Issued in Shared Mode.	N/A	N/A	N/A	N/A	N/A
95	Invalid START Function (Modifier Byte Error)	START	Disk	Consecutive or Indexed	Input, IO, or Shared	START function modifier byte does not correspond to a valid START option.
95	Primary Key Value Was Changed when Rewriting an Indexed Record.	REWRITE	Disk	Indexed or Alternate Indexed	IO or Shared	Attempted to change the value of the Primary Key while rewriting a record.

FILE STATUS FOR UFBERRAD RETURN (cont'd)

FS CODE	MEANING	FUNCTION REQUEST	DEVICE	FILE ORGANIZATION	MODE	CAUSE
96	Invalid Disk Address Detected.	N/A	N/A	N/A	N/A	Error usually not caused by user program. Error can occur for invalid disk address in the extent list (possibly caused by incorrect device arrangement at SYSGEN). This file status is returned only if the IOSW indicates invalid command or data address. Under RAM, DMS supplies the buffer area and command; thus, FS = 96 is a rare error under RAM.
96	Write Operation Attempted on Write-Protected Disk.	N/A	N/A	N/A	N/A	An attempt was made to write to an open write-protected diskette (this can occur if a user remounts a diskette changing it to write-protected but not using the MOUNT command).
96	Invalid Data Area Location or Alignment (IO Command error).	READ, REWRITE, or WRITE	N/A	N/A	Input, IO, or Output (Block Level)	Data area location is invalid or alignment is not on a page boundary. Data area location is checked with data area length to ensure that only the stack, static area, or buffer area is being used.
96	Same as 96 above.	READ or REWRITE	Workstation	N/A	IO	Invalid data area location or alignment (word alignment required.
97	Invalid Length when Rewriting Variable Length Record.	REWRITE	Disk	Indexed	N/A	Invalid length is indicated when attempting to rewrite a variable-length record whose length is longer than the value established in UFBRECSIZE.
97	Same as 97 above.	REWRITE	Disk	Consecutive	IO	Invalid, cannot change record length of a consecutive file.
97	Invalid Length Supplied when Writing Variable-Length Record.	WRITE	N/A		Output, Shared, IO, or Extend	Invalid length is indicated when attempting to write a variable-length record whose length is greater than the value established in UFBRECSIZE.

FILE STATUS FOR UFBERRAD RETURN (cont'd)

FS CODE	MEANING	FUNCTION REQUEST	DEVICE	FILE ORGANIZATION	MODE	CAUSE
97	Invalid Record-Prefix Found in Variable-Length Record.	N/A	N/A	N/A	N/A	Error encountered while DMS is attempting to extract a variable-length record from its buffer. Error should not normally be encountered by the user.
97	Invalid Length Specified for IO Operation.	N/A	Printer, Tape, or Workstation	N/A	N/A	Length specified is not valid for the device. For the printer, length is invalid if it equals zero or is larger than the length specified at SVC OPEN. For the workstation, length is invalid if data length and starting row cause screen overflow. For tape, length is invalid if a long block or a short block (with non-integral number of records) is read.
98	Invalid Alternate Tree Mask Supplied on Write or Rewrite Function.	WRITE or REWRITE	Disk	Alternate Indexed	Output, IO, or Shared	Alternate key mask references a nonexistent alternate key. For Write or Rewrite, the user-supplied mask must indicate valid ALT-trees and the alternate key fields must fall within the record; otherwise, FS=98 is returned. NOTE: A mask of zero is always valid.
99	Invalid Format Found for Current File Block.	N/A	N/A	N/A	N/A	A block within a variable-length record file has an invalid prefix, a VLEN record has an invalid prefix, or a compressed record has an invalid format when expanded.

All file status codes peculiar to ADMS fall within the range of FS=40 to FS=49. These codes and their meanings are as follows:

FS CODE	MEANING	FUNCTION REQUEST	DEVICE	FILE ORGANIZATION	MODE	CAUSE
41	Attempt to Update an ADMS File while Not in Transaction State.	WRITE, REWRITE, DELETE	Disk	ADMS	N/A	User error in order of operation.
42	ADMS Mapping Control Table (MCC) Error Detected.	READ, WRITE, REWRITE, DELETE	Disk	ADMS	N/A	Internal error in MCC table; unable to perform data mapping.
43	ADMS Data Item Integrity Violated.	WRITE, REWRITE	Disk	ADMS	N/A	Data item failed Integrity Check (ICK).
44	Unable to Log Record Image for ADMS Recovery.	WRITE, REWRITE	Disk	ADMS	N/A	DTI SVC failed while attempting to map the record image to sharer for logging to audit file.
45	View Access Block (VAB) Information Not Consistent with UFB.	N/A	Disk	ADMS	N/A	VAB improperly set up (System error).
46	Unable to Do Operation, File Previously Crashed.	N/A	Disk	ADMS	Shared	A crash condition has been detected for the data base. No request except CLOSE will be honored.
47	Unable to Perform ADMS Data Conversion Due to Invalid Data Format.	WRITE, REWRITE	Disk	ADMS	N/A	Unable to perform data mapping due to invalid data. Example: attempt to convert signed ASCII to binary but sign byte is invalid.
48	Entire ADMS File Must Be Held for Shared Update with Unique ALT Key.	WRITE, REWRITE, DELETE	Disk	Alternate-Indexed ADMS	Shared	Error in user's hold logic. ADMS file with recovery has ALT index with no dups; user failed to hold entire file when making update.

Summary of Changes

For 1st Edition of VS Operating System Services*

Type	Description	Affected Pages
NEW FEATURES	. Labelled tape support: - Changes to DMS functions and OPEN and CLOSE	305-315
	. Telecommunications Support - New Macro: TCOPTION - Changed Macros: READ START, UFBGEN, WRITE - Changed SVCs: OPEN CLOSE, CHECK	 127 100, 121, 131, 142 177, 182, 199
	. New Macrointructions - DISMOUNT - HALTIO - MOUNT - READFR - SET - TCOPTION	 63 77 87 104 116 127
DOCUMENTATION CHANGES	. Documentation of the following macroinstructions is not included in this manual: - FREEMEM - GETMEM - LOW - SEND - WAIT	See VS System Development Guide (800-1108SD-01)
	. Documentation of the following SCVs is not included in this manual: - WAIT - SEND - FIX/UNFIX - GETMEM - FREEMEM - DTI - SEIZE/RELEASE - GETDISK - FREEDISK - CREATFDR - UPDATFDR - PLEASE - MWAIT	See VS System Development Guide (800-1108SD-01)

* This manual replaces the VS System Programmer's Guide. The summary of changes refers to differences between this manual and the last edition of the System Programmer's Guide (Pub. #800-1103SP-03).

Type	Description	Affected Pages
	. Documentation of the following control blocks is not included in this manual: - CMSG - DBTB - DPT - ETCB - FLUB - FMSG - IORE - MCB - OFB - PFB - PFSA - PFT - PRB - PT - PXE - RMSG - STMB - SVCT - TCB - TQEL - UCB - VCB	See VS System Development Guide (800-1108SD-01)
TECHNICAL CHANGES	. Changes to EXTRACT Macro and SVC . Changes to HALTIO SVC . Revisions and clarifications of DMS messages . Changes to EXTRD control Block	64, 226 194 348 - 354 156
EDITORIAL CHANGES	. Miscellaneous Editorial Changes	

SUMMARY OF CHANGES FOR 2ND EDITION OF
VS OPERATING SYSTEM SERVICES MANUAL

TYPE	DESCRIPTION	AFFECTED PAGES
RELEASE 4 SUPPORT CHANGES	CEXIT	53 through 54.1
	CHECK	55 through 56.1
	CLOSE	57
	CREATE	58
	DISMOUNT	63, 63.1
	EXTRACT	64, 67 through 69.1
	FMTLIST	70
	HALTIO	77 through 73.1
	KEYLIST	79 through 81.1
	MOUNT	87 through 89
	READVTOC	106.1 through 106.4
	REGS	107
	SET	116 through 119
	SUBMIT JOB	126.1 through 126.6
	RETURN CODES	126.7, 126.8
	XMIT	148, 148.1
EDITORIAL CHANGES	IORE (I/O Request Element)	171.1, 171.2
	OFB (Open File Block)	171.3, 171.4
	UFB (User File Block)	173.1 through 173.12
	READVTOC SVC	204, 205
	MOUNT	231
	STATIC BLOCK	326, 327, 328
	DATANAME SUBBLOCK	338

Summary of Changes
for the
VS Operating System Services Addendum
(800-11070S-02.01)

TYPE	DESCRIPTION	PAGES
NEW FEATURE	Index	355-367
TECHNICAL CHANGES	Chapter 2	
	External Names in Code	29, 30, 31, 32, 32.1
	Chapter 4	
	BCTGEN Buffer Pool	49
	CEXIT	54
	CHECK Operand	55, 56.1
	CXT Suffix	59
	DELETE Syntax	60
	DESTROY Operand	62
	EXTRACT	65, 69
	FREEBUF Return Code	72
	GETBUF	73
	GETPARM Operand	74, 76
	MSGLIST Operand	90
	OPEN Syntax and Operand	91, 92
	RENAME Syntax, Operand, Return Code	108, 109 109.1
	REWRITE Coding Example	113
	SCRATCH Syntax	114
	SET Operand	117
	SUBMIT Syntax	126.1
	XIO Coding Example	147
	Chapter 5	
	OFB Fields and Flag	171.3, 171.4
	TPLAB	172

TYPE	DESCRIPTION	PAGES
TECHNICAL CHANGES	UFB Open Format Error	173.4
	WP and Workstation	173.5, 173.7
	UFBFPCLASS & UFBFILENAME	173.7
	UFBXCODE	173.10
	WP File Control	173.10
	7-Track Tape	173.12
	Chapter 6	
	OPEN File & Volume Names	177
	UFBFS2XFORMAT	180
	XIO Halt I/O Queue	185
	Return Codes	186
	GETBUF Return Codes	192
	FREEBUF Return Code	193
	HALTIO Input	194
	CHECK Input	199
	Unsolicited Interrupt	200
	Function 6	201
	READVTOC Return Code	203
	GETPARM Uppercase Alphanum	213
	Alphanumeric Limited	214
	RENAME Return Code	223
	EXTRACT OS Version Number	227
	EXTRDIDTAPEVOL	229
	MOUNT Input Parameter	230
	Return Code	231
	SET Procedure Keyword	239
	XMIT NOWAIT & OTHERTASK	240
	CREATE Privileged & GETMEM	241
	CEXIT NODEBUG & HELP	243
	DISMOUNT GETMEM	244
EDITORIAL CHANGES	Miscellaneous	53, 58, 74, 76, 83, 90, 93, 97, 98, 106.2, 106.3, 106.4, 109, 109.1, 110, 113, 114, 115, 126.3, 126.4, 126.5, 126.6, 173.10, 202, 203, 222, 241, 327
	Editorial Changes	

INDEX

Macroinstructions and SVCs

Alphabetical List

WANG

Customer Comment Form

VS OPERATING SYSTEM
Title SERVICES MANUAL

Publications Number 800-11070S-03

Help Us Help You . . .

We've worked hard to make this document useful, readable, and technically accurate. Did we succeed? Only you can tell us! Your comments and suggestions will help us improve our technical communications. Please take a few minutes to let us know how you feel.

How did you receive this publication?

- ☐ Support or Sales Rep
- ☐ Wang Supplies Division
- ☐ From another user
- ☐ Enclosed with equipment
- ☐ Don't know
- ☐ Other _____

How did you use this Publication?

- ☐ Introduction to the subject
- ☐ Classroom text (student)
- ☐ Classroom text (teacher)
- ☐ Self-study text
- ☐ Aid to advanced knowledge
- ☐ Guide to operating instructions
- ☐ As a reference manual
- ☐ Other _____

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR	VERY POOR
Technical Accuracy — Does the system work the way the manual says it does?	☐	☐	☐	☐	☐
Readability — Is the manual easy to read and understand?	☐	☐	☐	☐	☐
Clarity — Are the instructions easy to follow?	☐	☐	☐	☐	☐
Examples — Were they helpful, realistic? Were there enough of them?	☐	☐	☐	☐	☐
Organization — Was it logical? Was it easy to find what you needed to know?	☐	☐	☐	☐	☐
Illustrations — Were they clear and useful?	☐	☐	☐	☐	☐
Physical Attractiveness — What did you think of the printing, binding, etc?	☐	☐	☐	☐	☐

Were there any terms or concepts that were not defined properly? ☐ Y ☐ N If so, what were they? _____

After reading this document do you feel that you will be able to operate the equipment/software? ☐ Yes ☐ No ☐ Yes, with practice

What errors or faults did you find in the manual? (Please include page numbers) _____

Do you have any other comments or suggestions? _____

Name _____ **Street** _____

Title _____ **City** _____

Dept/Mail Stop _____ **State/Country** _____

Company _____ **Zip Code** _____ **Telephone** _____

Thank you for your help.

Printed in U.S.A. 14-3140 3-82-5C

WANG

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

WANG LABORATORIES, INC.
CHARLES T. PEERS, JR., MAIL STOP 1363
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Fold

Cut along dotted line.

WANG

ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851
TEL. (617) 459-5000
TWX 710-343-6769, TELEX 94-7421

Printed in U.S.A.
800-1107OS-03
8-82-8M