WANG

VS

Operating System Services Reference

Release 7 Series

IMPORTANT USER NOTICE

The VS Operating System Services Reference is a controlled release draft, intended for use with controlled Release 7.06 of the VS Operating System. This draft describes certain Release 7.10 Operating System features that will not be available until the general release of the product. At this time, Multivolume files, volume sets, and the Resource Sharing Facility are not included in the controlled release 7.06 version of the VS Operating System.

VS
Operating System Services Reference
Release 7 Series

1st Edition — October 1985

715-0423

WANG

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 the product was acquired, nor increases in any way Wang's liability to the customer. In no event shall Wang or its subsidiaries be liable for incidental or consequential damages in connection with or arising from the use of the product, the accompanying manual, or any related materials.

SOFTWARE NOTICE

All Wang Program Products (software) are licensed to customers in accordance with the terms and conditions of the Wang Standard Software License. No title or ownership of Wang software is transferred, and any use of the software beyond the terms of the aforesaid license, without the written authorization of Wang, is prohibited.

WARNING

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device, pursuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

PREFACE

The VS Operating System Services Reference provides users of the VS operating system with detailed reference information on what the system services are and how to use them. The VS system services allow experienced assembler programmers to use operating system routines to control the execution and interaction of programs.

Intended Audience

This manual is intended for system and application programmers who are programming in the assembler language. It is assumed that the user is familiar with the VS operating system and is an experienced assembler language programmer. For an overview of the VS operating system, refer to PART III of this manual.

PART I provides summary information on the use of the sytem services.

- Chapter 1 introduces the topic, defines the categories of systems services and lists those available.

- Chapter 2 describes how to call system services.

PART II provides detailed reference information on each system service.

- Chapter 3 describes detailed reference information on the system services that utilize the JSI instruction as well as associated macroinstructions. The descriptions are presented in alphabetical order for ease of reference. Examples of using some of these system services are provided.

- Chapter 4 contains descriptions of the services that are invoked by issuing an SVC instruction as well as associated macroinstructions. The descriptions are listed in alphabetical order. This chapter also contains a description of the control blocks that are of interest to the user.

PART III provides an overview description of the VS operating system.

- Chapter 5 discusses the user program and concepts relating to the development of programs within the VS operating system environment.

- Chapter 6 describes the VS operating system concepts that aid the user in understanding how the operating system manages the resources of the computing system.

- Appendix A includes information concerning program file structure and processing.

- Appendix B is a glossary to be used as a quick reference of terms while using the manual.

Throughout this manual, the following notation conventions are used:

[ ] Brackets indicate that the enclosed parameter is optional.

{ } Braces indicate that a selection is to be made from the enclosed list of elements. If a default value is supplied, it is indicated by an underscore. If the element is not coded, the underscored default value is assumed.

... An ellipsis indicates that the element may be repeated.

UPPERCASE Syntax elements presented in uppercase characters must be supplied exactly as shown in the statement.

lowercase Syntax elements presented in lowercase characters indicate elements to be supplied by the programmer.

All punctuation marks, such as commas, parentheses, or equal signs, must be coded as shown. In the syntax descriptions, the assembly rules for coding labels, variable names, and register specifications apply.

ASSOCIATED PUBLICATIONS

The following publications provide information that is helpful to the assembler language programmer:

- VS Assembler Language Pocket Guide (800-6203AP)
- VS Assembler Language Reference (800-1200AS)
- VS DMS Reference (800-1124)
- VS DMS/TX Reference (800-1128)
- VS Operating System Services Pocket Guide (715-0424)
- VS Principles of Operation (715-0422)
- VS Program Development Tools Reference (715-0884)
- VS Programmer's Introduction (715-0417)

OPERATING SYSTEM SERVICES REFERENCE MANUAL
RELEASE 7.10

CONTENTS

PART I USING SYSTEM SERVICES

CHAPTER 1 INTRODUCTION TO SYSTEM SERVICES

CHAPTER 2 CALLING SYSTEM SERVICES

PART II SYSTEM SERVICES DESCRIPTION

CHAPTER 3 JSI-TYPE SYSTEM SERVICES AND RELATED MACROINSTRUCTIONS

CONTENTS (continued)

CONTENTS (continued)

TABLES

FIGURES

CONTENTS (continued)

CHAPTER 1
INTRODUCTION TO SYSTEM SERVICES

1.1 OVERVIEW

System services are software routines that are part of the operating system. They perform functions that most user and application programs, as well as the operating system itself, commonly perform. Although most system services are used primarily by the operating system, some services are available for use by application programs.

Assembly language programmers can use system services to efficiently control the execution and interaction of programs.

- The macroinstructions save programming time because the necessary code has already been written.

- The macroinstructions save debugging time because they have already been tested and debugged.

- If a change in the supervisor call or a data structure occurs, the macroinstructions are automatically updated. The programmer only needs to reassemble the program to incorporate the changes.

For example, the Security Logging facility records security-related system events in a log file. To meet specific security needs, the programmer can write a program that calls the CNTRLOG system service to enable or disable logging, and to retrieve logging information.

There are two types of system services:

- JSI-type system services -- these services use the JSI (jump to subroutine) instruction to call an individual service routine. These services can be called from an assembly language or high-level language program at run time.

- SVC-type system services -- these services use the SVC instruction to call an individual service routine. These services can be called from an assembly language program only.

This manual describes both types of system services and related information necessary for using them most efficiently in an assembly language program. Chapter 3 covers the JSI-type system services and related macroinstructions descriptions. Chapter 4 covers the SVC-type system services and related macroinstructions descriptions.

1.2 SUMMARY OF SYSTEM SERVICES

Both JSI-type and SVC-type system services that are available to user programs are grouped into the following categories, according to the function they perform:

- Program services, including program initiation; program termination; timing; interrupt handling; and data structure maintenance.
- I/O services, including granting resources to requesting tasks and driving peripheral devices such as printers, tape and disk drives, and terminals.
- Memory management services, including dynamic allocation of heap storage (buffers); creating and accessing files in memory that contain code or data that can be shared.
- Communication and synchronization services, including transmitting commands and data from one task to another and sharing data between tasks.
- File services, including managing files (opening, closing, deleting and renaming).
- Security services, including protecting data structures and tasks; and ensuring privacy to users.

Sections 1.2.1 through 1.2.6 summarize the system services according to these functional categories. Each service is grouped in a category for organization only. A service can be used to fulfill other functions as needed by a particular program.

1.2.1 Program Services

Program services include functions such as program initiation, termination, and program resource management. The LINK and UNLINK services accomplish program initiation and termination. The command processor initiates user programs when the user issues a run request at the workstation. The user program can then link to other user programs by invoking LINK. Each time LINK is invoked, a new link level is created. Each link level is represented by a data structure called the program file block, which keeps track of program information during the course of program execution, and a LINK save area which is built on the modifiable data area stack. LINK performs such functions as allocating system control blocks used to monitor the called program, initializing the modifiable data area static area for the called program, and transferring control to the new program.

Once the user program has been executed, the RETURN macroinstruction returns control to the UNLINK service. In this way, all operations performed by LINK are reversed, and the calling program resumes execution. UNLINK performs such functions as closing all remaining open files, releasing devices which were reserved at the current link level, deallocating system control blocks, cleaning up stack data to the original address before the call to the program, and returning control to the command processor or the previous link level.

Abnormal Termination of a Program

Abnormal termination of a program may occur in response to one of the following actions:

- The user presses the HELP key and requests abnormal termination by pressing the PF key to cancel the program.

- The program issues a CANCEL or enters the Debugger as a result of a program check, and the user requests abnormal termination in response to the Debugger prompt.

- The program issues a CANCEL SVC, or a program check was issued and a CEXIT SVC with the NODEBUG or DUMP option was previously set.

The abnormal termination routines provide support for system resource retrieval in the event of a program malfunction or user-selected early termination. CANCEL is invoked by the system or user when a nonrecoverable error condition has occurred. An error message describing the type of error may be coded with the call to CANCEL. Through the use of the CEXIT service, a user program can cancel processing at a certain link level and receive control directly.

Table 1-1 summarizes the program services. For detailed instructions on using these services, refer to Chapters 3 and 4.

Table 1-1. Program Services and Related Macroinstructions

Service Name	Function
CALL	Provides linkage information to transfer control to another routine.
CANCEL	Cancel a program in the event of an uncorrectable program error.
CEXIT	Cancel or set link level parameters which specify the way a program handles error conditions.
CHARGEN	Generate 8 by 8 space characters for each character entered.
CXT	Symbolically reference the information returned to a program's cancellation-intercept routine.
DFB	Describes the data structure of a document file block (DFB).
ENDLOCAL	Terminates the automatic generation of local symbol names started by LOCAL.
EXTRACT	Extracts data from system control blocks for use in programs.
EXTRD	Describes the data structure which stores the output from EXTRACT.
FMTLIST	Generates the control block for input to the GETPARM and PUTPARM services.
GETPARM	Solicits information from users or from procedures.
KEYLIST	Generates a data structure which is used by GETPARM to store the response to GETPARM.
LINK	Initiates the execution of another program from within the currently active program.
LINKPARM	Supplies parameters to another program's GETPARM, cleans up data structures created by LINKPARM's PUT option; and allows the calling program to access changed parameters or previously created parameters.
LNKB	Describes the link return block (LNKB) used with LINK.

Table 1-1. Program Services and Related Macroinstructions (continued)

Service Name	Function
LOCAL	Automatically generates local symbol names.
LOGOFF	Generates code to issue logoff by program request.
MSGLIST	Generates a data structure to use with the GETPARM's MSG parameter and CANCEL.
PCEXIT	Allows execution of a user-written exception handling routine for user-selected exceptions.
PROCINFO[a]	Provides user programs with information related to a specific process or task.
PUTPARM	Enables a program to supply parameters to a GETPARM issued by another program.
REGS	Equates register numbers with the standard symbolic names used by other macroinstructions.
RETURN	Exit conditionally from a program to the system for a standard termination.
SET	Sets default values for task related parameters.
SYSERROR	Establishes symbolic names with their numeric codes for common system error conditions.
TCOMPLET[a]	Allows a parent task to check on the completion of its child task.
TINVOKE[a]	Allows a running program to create a child task.
TKILL[a]	Allows a parent task to force a child task and all of the descendants into CANCEL and LOGOFF.

[a] JSI-type system service.

1.2.2 I/O Services

User programs can use the I/O services to manage peripheral devices such as printers and tape drives. I/O services include managing the physical devices used during an I/O (i.e. mounting and dismounting a volume) and managing the resources associated with I/O (holding and releasing telecommunications devices). I/O services can be used to complete these tasks while a program is executing.

The I/O services perform the following initiation and completion routines for the operating system:

- Manage workstation screen display and interaction with the user.
- Open and close channels for telecommunication devices.
- Load microcode to devices.
- Reserve and release telecommunication devices, lines and peripherals.

Table 1-2 summarizes the I/O services. For detailed instructions on using these services, refer to Chapters 3 and 4.

Table 1-2. I/O Services and Related Macroinstructions

Service Name	Function
CHECK	Checks for an occurrence of an event (I/O operation, timing interval expiration, message to be sent, PF key, unsolicited interrupt, TC I/O, semaphore wait, session ID, mailbox) or combination of events.
DISMOUNT	Requests dismount of disk or tape volume.
HALTIO	Stops an input/output operation started by XIO.
LOADCODE	Loads microcode into a processor or device.
MOUNT	Issues a mount request for a disk or tape volume.
READVTOC	Reads the volume table of contents (VTOC) of a disk.

Table 1-2. I/O Services and Related Macroinstructions (continued)

Service Name	Function
TPLAB	Describes file header, trailer, and end-of-volume labels for a magnetic tape.
TPLAB2	Describes secondary file header, trailer, and end-of-volume labels for a magnetic tape.
VOL1	Describes the standard volume label for disk or magnetic tape.
VOLINFO[a]	Extracts system information on a specific volume.
VSETINFO[a]	Extracts volume information on volume sets.
XIO	Manages the physical I/O operation.

[a] JSI-type system service.

1.2.3 Memory Management Services

The memory management services provide buffer management, memory protection and memory mapping functions for VS systems. These services manage buffers and heap storage areas. Also, program and data files can be mapped into a task's virtual address space.

Table 1-3 summarizes the memory management services. For detailed instructions on using these services, refer to Chapters 3 and 4.

Table 1-3. Memory Management Services and Related Macroinstructions

Service Name	Function
FREEBUF	Releases a buffer area allocated by GETBUF.
FREEHEAP	Releases heap storage area allocated by GETHEAP.
GETBUF	Allocates a buffer area on a 2048-byte (one page) boundary.
GETHEAP	Dynamically allocates system storage, in any size block, independent of the system stack.
MSMAP[a]	Maps program and data files into a task's virtual address space.
MSUNMAP[a]	Unmaps a file from a task's virtual address space.

[a] JSI-type system service.

1.2.4 Communication and Synchronization Services

The communication and synchronization services provide a method for tasks to cooperate with one another to perform complex functions. This typically involves transmitting commands and data from one task to another or sharing data between tasks. Synchronization operations control task access to common or shared data areas. This technique prevents a task from destroying the integrity of shared data by simultaneously updating the same data record or reading a record before another task has finished updating it

To control task execution, the VS operating system uses semaphores that are not available to user-level code. Semaphores act like gates into critical areas of software to protect shared data or I/O.

To control access to shared data in user-level code, the VS operating system provides the User Synchronization facility, a fast, simple synchronization technique. System services that can be called from a user program allow a user to create, delete and use a synchronization object to coordinate the access to shared data. The synchronization object is probably used most often for resource control, that is to update a data base or to access a specific piece of code. However, it can be used to satisfy other application needs as well.

Clock Interruptions

The VS central processor supports two timer-related values which are stored in control registers: the time-of-day clock and the clock comparator. The time-of-day clock is a value, contained in one or two control registers (depending on VS system), that is incremented periodically, independent of central processor activity. The clock comparator is a value, contained in one or two control registers (depending on VS system), that is continuously compared with the time-of-day clock. Whenever this comparison finds the time-of-day clock to be equal to or greater than the clock comparator, a clock interrupt is made pending. Any task running under the operating system may request interval timing services.

For more information on communication and synchronization, refer to Chapter 6.

Table 1-4 summarizes the communication and synchronization services. For detailed instructions on using these services, refer to Chapters 3 and 4.

Table 1-4. Communication and Synchronization Services and Related Macroinstructions

Service Name	Function
CHECK	Checks for an occurrence of an event (I/O operation, timing interval expiration, message to be sent, PF key, unsolicited interrupt, TC I/O, semaphore wait, session ID, mailbox) or combination of events.
CREATE	Creates an intertask message receipt port.
DESTROY	Deletes an intertask message receipt port.
IPCB	Describes the interprocessor control block (IPCB).
IPCLOSE	Closes a specified number of telecommunications devices that were opened with IPOPEN.
IPOPEN	Opens specified telecommunications devices for I/O between the operating systems and the data link processor (DLP).

Table 1-4. Communication and Synchronization Services and Related Macroinstructions (continued)

Service Name	Function
RECEIVE	Initiates a data reception operation between the operating system and the data link processor (DLP).
RESETIME	Cancels an interval timing request previously established by SETIME which has not been the subject of a CHECK INTERVAL or previous RESETIME.
SBREAK[a]	Removes a task that is holding a synchronous object and gives the object to the task that issued the break synchronization call.
SCREATE[a]	Creates a data structure that controls the use of a shared resource.
SDELETE[a]	Marks a synchronous object for delete, thereby disallowing any new waiters to enter the queue.
SENTER[a]	Issues a request to gain control of the synchronization object to use the resource.
SETIME	Sets a timer interval for the issuing task to expire at the time specified, or after the number of 1/100 second units specified.
SEXIT[a]	Releases the caller from control of the resource, and activates the next waiter.
SUBMIT	Transfers files from one system to another over WangNet. It also submits files for printing.
TCOPTION	Sets the TC stream options in the user file block (UFB).
TRANSMIT	Initiates an I/O operation directed to the DLP on the addressed communication channel device.
UNITRES	Reserves and releases exclusive use of telecommunications devices, lines, and peripheral processors.
XMIT	Sends a message to a specified intertask message port.

[a] JSI-type system service.

1.2.5. File Services

File services support many file management routines including file resource allocation, file information update and retrieval, DMS file transaction, and file open, close, delete and rename.

Table 1-5 summarizes the file services. For detailed information on using these services, refer to Chapter 4.

Table 1-5. File Services and Related Macroinstructions

Service Name	Function
AXD1	Allows symbolic reference to the alternate descriptor block (AXD1) which describes the alternate index structure of an indexed file.
AXDGEN	Generates the AXD1 block.
BCE	Describes the buffer control entry (BCE) contained in the buffer control table (BCT).
BCTBL	Describes the buffer control table (BCT).
BCTGEN	Generates a buffer control table for use in buffer pooling.
BEGTRANS	Marks the beginning of a DMS transaction.
CLOSE	Closes a file.
DELETE	Deletes the last record read from an indexed file on disk.
DEXIT	Provides a deadlock exit from DMS/TX.
FDR1	Describes the file descriptor record block, format 1 (FDR1), which contains the attributes of the file and the first three extents of single volumes.
FDR2	Maps symbol names to the file descriptor record block, format 2 (FDR2), which describes up to 10 additional extents to a file for a single volume file; up to nine additional extents for a multivolume file.
FDR3	Maps symbol names to the file descriptor record block, format 3 (FDR3), which contains information on files on volume sets.

Table 1-5. File Services and Related Macroinstructions (continued)

Service Name	Function
FREEALL	Frees all resources acquired through the sharing task.
FREESHR	Releases all of user's resources acquired through the sharing task.
FREEXRTS	Releases extension rights acquired through GETXRTS (DMS function).
GETXRTS	Acquires more resources while already holding resources (DMS function).
OPEN	Opens a file.
PROTECT	Updates protection information (protection class, owner of record, expiration date) for a disk file or a library of disk files.
READ	Reads a record from a file or device supported by DMS.
READFDR	Locates a disk file on a specified volume and copies its FDR1, FDR2, or FDR3 blocks into memory.
READVTOC	Provides a disk volume table of contents (VTOC) information.
RENAME	Renames a disk file or library.
REWRITE	Rewrites a record to a file or device.
ROLLBACK	Undoes a DMS/TX transaction.
SCRATCH	Deletes a disk file or library from a volume.
SETRECOV	Attaches or detaches a file with recovery blocks to a DMS/TX database, or clears a crash status.
START	Start file processing in a specified mode or at specific record.
START HOLD/RELEASE	Requests a hold or release on resources in a data file.

Table 1-5. File Services and Related Macroinstructions (continued)

Service Name	Function
SUBMIT	Transfers files from one system to another using WangNet. It also submits files for printing.
UFB	Describes the user file block (UFB).
UFBGEN	Generates the user file block (UFB) with specified fields initialized.
UPDATFDR	Updates existing FDR blocks.
WPCALL	Calls routines to do I/O on a word processing document.
WRITE	Writes the next consecutive record (consecutive or indexed files) or writes a specified record (indexed file).
WV46MAP	Maps the parameter list supplied to SUBMIT and provides information to use by SUBMIT with the PLIST option.

1.2.6 Security Services

The Security Logging facility tracks security related events that occur during system operation and stores this information in a log file. Security Logging not only provides a method of accountability for system use, but can also serve as an effective deterrent to security violations. Application programs can be written to control the Security Logging facility using the system services that support the facility. These services can be used only by System Administrators.

Table 1-6 summarizes the security services. For detailed information on using these services, refer to Chapters 3 and 4.

Table 1-6. Security Services and Related Macroinstructions

Service Name	Function
CNTROLOG[a]	Communicates control information to the operating system security logging task.
LOGR	Generates a DSECT which defines all the fields found in a security logging system PUTLOG record, their identifiers, and the event types and subtypes.
PROTECT	Updates protection information (protection class, owner of record, and/or expiration date) for a disk file or a library of disk files on a volume.
PUTLOG[a]	Inserts a record into the system security event logging database file.

[a] JSI-type system service.

CHAPTER 2
CALLING SYSTEM SERVICES

2.1 OVERVIEW

This chapter describes how to call both the JSI-type services and the SVC-type services. Examples of using the JSI-type services are provided in Section 3.3. As part of those examples, SVC-type services are also used. Refer to Chapter 6 for a detailed description of how the program stack is handled for the JSI-type and SVC-type services.

2.2 CALLING THE SYSTEM SERVICES

The system services that use the JSI instruction do not use supervisor calls (SVC) to perform the service. As a result, there is more flexibility in their use because they could be loaded to any free space in memory, whereas SVC-type services are loaded at a defined location.

NOTE

The JSI (jump to subroutine) instruction operates in the same manner as the JSCI (jump to subroutine on condition indirect) instruction, except that the jump to the specified subroutine is always made. No conditions have to be met. The stack for the JSI instruction is handled the same way as the stack is handled for the JSCI instruction. Refer to Chapter 6 for more information on the JSCI instruction.

The macro definitions of the JSI-type services are stored in @MACLIB@ on the system volume. The executable code of the JSI-type services are part of a shared subroutine library, called @SYSSERV on the system volume. To call one of these services, the Linker is used. The shared subroutine library must be enabled (enter YES next to the field SHAREDSL for the prompt "Create a SHARED subroutine library?" on the Linker OPTIONS screen). An alias is a name of up to 40 characters assigned to each shared subroutine library. The alias for the JSI-type system services shared subroutine library is @SYSSERV. @SYSSERV is entered on the SSLALIAS screen. At runtime, the address of the service is resolved, the routine is called and run. It does not become part of the resident code of the program. Refer to Section 3.3 for examples on using these system services. Refer to VS Linker and Symbolic Debegger Reference for more information on using the Linker.

The SVC-type system services are located in the system library @MACLIB@ on the system volume. These services are a resident part of the operating system code. They are accessed when the program is run. Refer to VS Principles of Operation for more information on the SVC instruction.

The assembly language code for calling either type of service (JSI or SVC type) is the same. The name of the macro is entered, along with any necessary parameters, as follows:

```
[label] name_of_service [parameter], [parameter], ...
```

An example of a call for the MSUNMAP system service (JSI-type) is as follows:

```
MSUNMAP RETURNCODE=RC,PATHNAME=PTH
```

An example of a call for the GETPARM system service (SVC-type) is as follows:

```
GETPARM FORM=SELECT,KEYLIST=CNTRL,MSG=MSG1,PFKEYS=(R10)
```

When printing out assembled program code that includes system services, the option of printing the expanded macro statements is available. PRINT GEN prints the expanded macro; PRINT NOGEN does not print the expanded macro. When PRINT GEN is specified, a "+" symbol precedes all program statements generated by the macro. For example, lines 69 through 73 of the following program section are the assembler-generated statements for the GETPARM system service:

```
68      GETPARM FORM=SELECT, KEYLIST=CNTRL, MSG=MSG1, PFKEYS=(R10)
69+     PUSH    0,R10             Push the PF key mask on the stack
70+     PUSHA   0,CNTRL           Put the KEYLIST address on the stack
71+     PUSHA   0A,MSG1           Put the MSG address on the stack
72+     MVI     (0(15),B'00010100'  Move in the GETPARM options byte
73+     SVC     20 (GETPARM)
```

2.3 RETURN CODES

When a system service has been completed, a return code is issued. The return code indicates the status of the operation:

- If the operation was successful, the return code is always zero.

- If there was an error, the return code is a nonzero value. The relevant values returned for each service are described in the detailed service-by-service descriptions.

All return codes and associated error definitions are maintained in the system file SYSERROR in @MACLIB@.

2.4 ASSEMBLY LANGUAGE CODING CONVENTIONS

This section describes two conventions to remember when assembling a program. For more information on conventions to use when coding a program, refer to the VS Assembly Language Reference.

The first statement in a program should be a CODE statement. This causes the source code following the statement to be part of the reentrant program section named by the label in the CODE statement. The syntax for the CODE statement is as follows:

```
label CODE          Not used; should be blank
```

The label is required and may have a maximum of eight characters. It is used as the external name of this reentrant program section. Entry symbols in code or static sections are also limited to eight characters.

If a static area is desired, the STATIC statement should be used. The assembler allows any number of these statements and allows initial values to be specified. The syntax for the STATIC statement is as follows:

```
label STATIC        Not used; should be blank
```

The label is required and may have a maximum length of eight characters. It is used as the external name of this static section.

2.5 REGISTER CONVENTIONS

There are 16 general 32-bit registers provided for the programmer's general use. The standard conventions for the use of these registers are as follows:

- R0 through R13 -- general use. However, R1 is used as a pointer to an argument list for use with some system services. Refer to the service-by-service descriptions.

- R14 -- references static area (refer to the programming example in Section 4.3.1).

- R15 -- Register 15 is the stack pointer (SP) and must always address the lowest location on the stack which contains usable data or into which data may be placed by any non-PUSH instruction. This convention must be followed in all programs.

Use the REGS macroinstruction (refer to Chapter 4) for establishing symbolic names for the general registers.

CHAPTER THREE
JSI-TYPE SYSTEM SERVICES AND RELATED MACROINSTRUCTIONS

3.1 OVERVIEW

This chapter describes macros for system services that require special linking procedures for their use. Chapter 4 discusses system services invoked by the SVC instruction.

The macroinstruction definitions are contained in individual files (identified by name) in the library, @MACLIB@, on the system volume. The assembler may access one or more of these files when processing a source program containing macro calls.

3.1.1 V Type Address Constants

For system services described in this chapter, each macro generates a V type address constant for the linkage table entry of the system service that it invokes. The V type constant implies that the label is an external reference whose address will be resolved later. There is no need to declare the label as external by coding an EXTRN statement.

3.1.2 Linking the System Services

The executable object code for each system service resides in a system shared subroutine library called @SYSSERV on @SYSTEM@ of the system volume. The code is linked into the user program by supplying @SYSSERV as the alias during the link procedure. Refer to the VS Linker and Symbolic Debugger Reference for further information on how to link in a shared subroutine library.

3.2 SYSTEM SERVICE DESCRIPTIONS

In the following sections, each system service description contains the following information:

- Syntax -- This section describes the format for coding a macroinstruction. The programmer must adhere to assembly language syntax rules as described in the VS Assembly Language Reference when coding the macroinstructions. Parameters for the call are listed in the reverse order in which they are pushed onto the parameter block. That is, the return code address is always the last parameter pushed. As the macro generates the code to push the parameters in the expected order, assembly language programmers may code the call to the service with the parameters in any order. However, high-level language programmers must respect the order shown in the syntax section of the system service description.

- Function -- This section describes the functions of the service.

- Parameter definitions -- This section describes in detail the parameters that may be used with the macro call, and their valid values. Unless otherwise stated, the argument to a KEYWORD is the address of the value, not the value itself. The address may be a register specification in parenthesis or an address expression. This section also describes whether the parameter is an input or output parameter and the parameter's data type.

- Return Codes -- This section lists the valid return codes for the system service. A return code of zero always indicates success. The SYSERROR macro in Chapter 4 is provided for standardization of user program error message. The return code section is omitted for macroinstructions that generate or describe system data structures.

- Example -- The section contains at least one coding example for each macro. Also included is the code generated when the macro is expanded and the static sections statements containing constant or storage declarations for the parameters.

Data Types

The data type descriptions are represented in PL/1 notation for easy interpretation by high-level language programmers. Table 3-1 is a conversion chart from PL/1 to assembly language.

Table 3-1. Data Type Conversion Table

PL/1	Assembler
Fixed bin(31,0)	DS F
Fixed bin(15,0)	DS H
Char(n)	DS CLn
Char(n) var[a]	DC H'n' DS CLn
Bit(n)	DS BL.n
Pointer	DS A(symbol)

[a] The char(n) var data type assumes that the first two bytes (halfword aligned) contain a count of the number of characters that follow. The variable n specifies the maximum number of characters that may follow.

Error Handling Routines

Some of the services have an additional optional parameter for specifying the entry point of an error handling routine. The syntax is: [,ERROREXIT=label]. When a service returns a code indicating a failure in the call and the ERROREXIT parameter is specified, the system transfers control to the address specified with the ERROREXIT parameter.

3.2.1 CNTROLOG - Control Logging of System Security Events

Syntax

```
[label]  CNTROLOG    RC=returncode
                    [,SETEVENTS=setevents]
                    [,RESETEVENTS=resetevents]
                    [,SETVIOLATION=setviolation]
                    [,RESETVIOLATION=resetviolation]
                    [,CONTROL=control
                    [,NEWLIB=newlibrary]
                    [,NEWVOL=newvolume]
                    [,GETEVENTS=getevents]
                    [,GETVIOLATIONS=getviolations]
                    [,GETSTATUS=getstatus]
                    [,ACTFILE=activefile]
                    [,ACTLIB=activelibrary]
                    [,ACTVOL=activevolume]
                    [,INACTFILE=inactivefile]
                    [,INACTLIB=inactivelibrary]
                    [,INACTVOL=inactivevolume]
                    [,SETALTVOL=setalternatevolume]
                    [,SETNRECS=setnrecs]
                    [,GETALTVOL=getalternatevolume]
                    [,GETNRECS=getnrecs]
```

Function

CNTROLOG communicates control information to the operating system security logging task. This service provides the following functions:

1. Start and stop logging up to 256 individual types of events.

2. Start and stop logging of attempted violations of up to 256 individual events.

3. Specify a new file name to be used for logging events or continuing using an already active file.

4. Return the types of events which are currently being logged, the types of events whose attempted violations are being logged, the status of the logging task, the volume, library and file of the active logging file, the volume, library and file of the inactive logging file.

Events consist of logon, logoff, file open and close, file rename, file delete, file attribute change, userlist change, program invocation, procedure invocation, background job initiation, DP print request, WP print request, mount and dismount commands, operator-user communications, system messages to the operator, attach/detach of disks and printers, acquire/release of workstations, system snapshot dumps, and attempted violations. See the LOGR macro for the event/bit definitions.

When starting the logging task with the new log file option, the system creates a file name that consists of the time and date of file creation. The caller specifies the library and volume with the NEWLIB and NEWVOL parameters. To obtain the file specification for a log file just closed as a result of a new log file request, specify the INACTFILE, INACTLIB and INACTVOL output parameters on the same call to CNTROLOG as the request to start a new log file. The system returns the file specification of the new file in the ACTFILE, ACTLIB and ACTVOL parameters.

The caller must have system administrator and operator privileges to perform the privileged functions of this service.

Parameter Definitions

Parameter Definition	I/O	Data Type
activefile	Output	char(8) var

Returns the name of the currently active log file. If used when opening a new log file (CONTROL=2), CNTROLOG returns the name of the newly created log file.

activelibrary	Output	char(8) var

Returns the name of the library of the currently active log file. If used when opening a new log file (CONTROL=2), CNTROLOG returns the library of the newly created log file.

activevolume	Output	char(8) var

Returns the name of the volume of the currently active log file. If used when opening a new log file (CONTROL=2), CNTROLOG returns the volume of the newly created log file.

control Input fixed bin(31,0)

Changes the state of the logging facility. A value of 3 means that logging is restarted, and is to continue using the same log file that was used the last time logging was active. If logging is already active, this is an invalid request, and the caller is notified. A value of 1 causes logging activity to terminate (the caller is notified if logging is not active). A value of 2 causes a new log file to be opened. If logging is active at the time of the call, then the current file is closed. If logging is inactive, then logging is started. Parameter restricted to privileged callers.

getalternatevolume Output char(6)var

Returns the name of the volume to be used if the primary volume cannot be used. Cannot be used with SETALTVOL.

getevents Output bit(256)

Returns the events which are being logged. Each bit represents an individual event. This parameter may not be used with the SETEVENTS or RESETEVENTS parameters.

getnrecs Output fixed bin(31,0)

Returns the value set by the last SETNRECS. This is the number used to set the initial extent size when opening a new log file. It cannot be used with SETNRECS.

getstatus Output fixed bin(31,0)

Returns the state of the logging facility. A value of 0 means that logging is inactive. A value of 1 means that logging is active. This parameter may not be used with the CONTROL parameter.

getviolations Output bit(256)

Returns the events whose attempted violations are being logged. Each bit represents an individual event. This parameter may not be used with the SETVIOLATIONS or RESETVIOLATIONS parameters.

inactivefile Output char(8) var

Returns the name of the log file just closed by the CONTROL=2 action. This parameter may only be used in conjunction when specifying CONTROL=2. Restricted to privileged callers.

inactivelibrary   Output   char(8) var

Returns the name of the library of the log file just closed by the CONTROL=2 action. This parameter may only be used when specifying CONTROL=2. Restricted to privileged callers.

inactivevolume   Output   char(8) var

Returns the name of the volume of the log file just closed by the CONTROL=2 action. This parameter may only be used when specifying CONTROL=2. Restricted to privileged callers.

newlibrary   Input   char(8) var

The name of the library in which the new log file is to be created. This parameter is only valid when specifying CONTROL=2. Restricted to privileged callers. Defaults to last library used.

newvolume   Input   char(8) var

The name of the volume on which the new log file is to be created. This parameter is only valid when specifying CONTROL=2. Restricted to privileged callers. Defaults to last volume used.

resetevents   Input   bit(256)

Determines the events which are no longer to be logged. Each bit represents an individual event. Bits set to 1 will correspond to events to be turned off (not to be logged). If both SETEVENTS and RESETEVENTS are specified at the same time, RESETEVENTS will be processed first. Restricted to privileged callers.

resetviolations   Input   bit(256)

Determines the events whose attempted violations shall no longer be logged. Each bit represents an individual event. Bits set to 1 will correspond to events to be turned off (not to be logged). If both SETVIOLATIONS and RESETVIOLATIONS are specified at the same time, RESETVIOLATIONS will be processed first. Restricted to privileged callers.

returncode   Output   fixed bin(31,0)

Code indicating the success or failure of the routine call.

setalternatevolume Input   char(6) var

The name of a volume to be used as an alternate volume when the primary volume cannot be used. Defaults to NEWVOL if no previous ALTVOL specified. Restricted to privileged callers.

setevents Input bit(256)

Determines the events to be logged. Each bit represents an individual event. Bits set to 1 indicate the events to be logged. Restricted to privileged callers.

setnrecs Input fixed bin(31,0)

The size of the initial extent of a new log file (in number of records). This number is used to get UFBNRECS. Restricted to privileged callers.

setviolations Input bit(256)

Determines the events whose attempted violations are to be logged. Each bit represents an individual event. Bits set to 1 will correspond to events to be logged. Restricted to privileged callers.

Return Codes

Code	Definition
@ERACC	Access denied.
@ERGETRSTEVENTS	Cannot do both getevents and resetevents on same CNTROLOG call.
@ERGETRSTVIOLS	Cannot do both getviolations and resetviolations on same CNTROLOG call.
@ERGETSETEVENTS	Cannot do both getevents and setevents on same CNTROLOG call.
@ERGETSETVIOLS	Cannot do both getviolations and setviolations on same CNTROLOG call.
@ERINACTNOTNEW	Cannot request inactivefile when not doing a newlog on CNTROLOG call.
@ERIOERR	I/O error.
@ERIPTYP	Illegal parameter type.
@ERLOGGINGON	Logging is already active.
@ERLOGINACTIVE	Logging is not active.

Code	Definition
@ERLOGNOTPRIV	Caller not authorized to log this event type.
@ERNEWLIBNOTNEW	Cannot specify newlib and control if control not = newlog.
@ERNOREPLY	No reply message from SYSTSK.
@ERSTATCNTRL	Cannot do both control and getstatus on same CNTROLOG call.
@ERUNPRIV	Unprivileged caller.
@ERWRONGMSG	Invalid message sent back by SYSTSK.
@ERGETSETALT	Cannot do both getalternatevolume and setalternatevolume on same CNTROLOG call.
@ERGETSETNRECS	Cannot do both getnrecs and setnrecs on same CNTROLOG call.

Example

```
          CNTROLOG  RC=RCODE,GETEVENTS=EVENTMAP,
                GETVIOLATIONS=VIOMAP,ACTFILE=LOGFILE,ACTLIB=LOGLIB,
                ACTVOL=LOGVOL,GETSTATUS=ONOFF,STATIC=(R14)
+         PUSHA 0,LOGVOL             . Volume for Active Log .
+         OI    0(15),X'80'          . Indicate Parameter List End .
+         PUSHA 0,LOGLIB             . Library for Active Log .
+         PUSHA 0,LOGFILE            . Filename for Active Log .
+         PUSHA 0,ONOFF              . Get Status .
+         PUSHA 0,VIOMAP             . Get Violations .
+         PUSHA 0,EVENTMAP           . Get Events .
+         PUSHA 0,0                  . (New Log File's Volume) .
+         PUSHA 0,0                  . (New Log File's Library) .
+         PUSHA 0,0                  . (Control parameter) .
+         PUSHA 0,0                  . (Reset Violations) .
+         PUSHA 0,0                  . (Set Violations) .
+         PUSHA 0,0                  . (Reset Events) .
+         PUSHA 0,0                  . (Set Events) .
+         PUSHA 0,RCODE              . Return Code .
```

```
+#CNTROLG STATIC                    . Section for PUTLOG VCON .
+         ORG   #CNTROLG            . Start the section ... .
+         DC    V(CNTROLOG)         . ... with the VCON .
+         CSECT                     . Rejoin current section .
+         L     1,=R(#CNTROLG)      . Address Static Section .
+         L     1,0(R14,1)          . Add Static Base .
+         PUSH  0,1                 . Enstack VCON Address .
+         LA    1,4(,15)            . Address Parameters .
+         JSI   0(,15)              . Call PUTLOG .
+         POPN  0,60                . Restore Stack .

                   .
          (Static Section)
                   .

 RCODE    DS F
 EVENTMAP DC BL.256'0'
 VIOMAP   DC BL.256'0'
 LOGFILE  DC CL8'        '
 LOGLIB   DC CL8'        '
 LOGVOL   DC CL6'      '
 ONOFF    DC F'0'
```

3.2.2 LOGR - System Security Logging Record Format

Syntax

```
[label]  LOGR  [NODSECT][,STORAGE={ NO}]
                                  {YES}
```

Function

This macro generates a DSECT which defines all the fields found in a security logging system PUTLOG record, their identifiers, and the event types and subtypes. It optionally allocates storage for a code section through the NODSECT parameter. The STORAGE parameter controls the amount of storage allocated for a code section (if NODSECT is specified) or the offsets shown in a DSECT. Also may be used in conjunction with the CNTROLOG macro for setting events.

Parameter Definitions

Parameter Definition	I/O	Data Type
NODSECT	Input	

Specifying NODSECT results in storage being allocated as part of the current code or static section. If not specified, the system generates a dummy section (with no storage allocation) showing offsets relative to the beginning of the section.

Parameter Definition	I/O	Data Type
STORAGE	Input	

STORAGE=YES sets the replication factor for each DS statement to one starting with LOGR$TYPE. STORAGE=NO sets the replication factor for each DS statement to zero. If NODSECT is specified, specifying STORAGE=YES generates storage for the total macro. Specifying NODSECT,STORAGE=NO generates storage for the shorter form of the macro.

Example

```
         LOGR
+LOGR    DSECT
+**********************************************************************
+*                                                                    *
+* This DSECT contains the definition of all fields found in a PUTLOG *
+* record, their identifiers, the event types, and the event          *
+* subtypes.  Within this DSECT, a labeling convention is used that   *
+* makes things easier to follow.  The convention is:  labels         *
+* with '#' in them refer to field identification numbers; labels with *
+* '$' in them refer to the value portion of the field; labels with   *
+* '@' in them refer to possible values for the field.  Another       *
+* convention is that identifiers for common fields start with the    *
+* number 255 and descend.  Identifiers for type-dependent fields     *
+* start with the number 1 and ascend.                                *
+*                                                                    *
+**********************************************************************

+LOGRBEGIN       DS   0F
+LOGRRECLENGTH   DS   H          Total length of record
+LOGRFIELD       DS   0X         The format of a field in a PUTLOG
+*                               record.
+LOGRFIELDID     DS   XL1        Contains the field identifier.
+LOGRFIELDLEN    DS   XL2        Contains the length of the field
+*                               value (does not include the ID or
+*                               length bytes).
+LOGRFIELDVALUE  DS   0XL256     Contains the value of the field.
+LOGRHDR         DS   0X         The common fields of a PUTLOG record
+LOGR#TYPE       EQU  255        The ID # of the event type field
+LOGR$TYPE       DS   0XL2       The description of the value portion
+*                               of the event type field
+LOGR#SUBTYPE    EQU  254        The record subtype
+LOGR$SUBTYPE    DS   0XL2
+LOGR#TIME       EQU  253        Timestamp
+LOGR$TIME       DS   0XL8
+LOGR#VIOLATION  EQU  252        Violation Flag byte
+LOGR$VIOLATION  DS   0BL2
+LOGR@ALERT      EQU  1          . Record represents an attempted
+*                               violation
+LOGR#CUID       EQU  251        User ID of the PUTLOG caller
+LOGR$CUID       DS   0CL8
+LOGR#CWS        EQU  250        Workstation used by the PUTLOG caller
+LOGR$CWS        DS   0CL8
+LOGR#CJOB       EQU  249        Job name used by the PUTLOG caller
+LOGR$CJOB       DS   0CL8
+LOGR#SUID       EQU  248        User ID of the subject of the PUTLOG
+LOGR$SUID       DS   0CL8
+LOGR#SWS        EQU  247        Workstation used by the subject of
+*                               the PUTLOG
```

```
+LOGR$SWS       DS   0CL8
+LOGR#SJOB      EQU 246          Job name of the subject of the PUTLOG
+LOGR$SJOB      DS   0CL8
+LOGR#STASK     EQU 245          Task ID # of the subject of the PUTLOG
+LOGR$STASK     DS   0XL2
+LOGREVENTDATA  EQU   *

+******************************************
+*      User Application Event Type      *
+******************************************
+LOGRUSER       EQU 0            User application event type
+LOGR#USERDATA  EQU   1          ID # for user data field
+LOGR$USERDATA  DS    0XL256     User data

+******************************************
+*      Logon Event Type                 *
+******************************************
+         ORG   LOGREVENTDATA
+LOGRLOGON      EQU 1            Logon event type
+LOGR#LOGONERR  EQU 1            ID # for logon error field
+LOGR$LOGONERR  DS   0XL1        Error code when logon is rejected
+LOGR@INVIDPSW  EQU 1            . Invalid user ID or password
+LOGR@MULTLOG   EQU 2            . User logged on elsewhere
+LOGR@UNBOPRD   EQU 3            . Unable to open or read user list
+LOGR@LOGINHIB  EQU 4            . Logon inhibited
+LOGR@WSRESTR   EQU 5            . User restricted from WS
+LOGR@GETMEM    EQU 6            . Getmem error
+LOGR@GETBLK    EQU 7            . Getblk error
+LOGR@SEG2SZ    EQU 8            . Segment 2 size error
+LOGR@NOLOGPROC EQU 9            . NO LOGON PROC FOR NOHELP USER
*******************************************
+*       Logoff Event Type               *
+******************************************
+         ORG   LOGREVENTDATA
+LOGRLOGOFF     EQU 2            Logoff Event Type
+LOGR#LREASON   EQU 1            ID # for Reason for Logoff field
+LOGR$LREASON   DS   0XL1
+LOGR@LNORMAL   EQU 0            . User - initiated (normal) logoff
+LOGR@LFORCED   EQU 1            . Forced Logoff
```

```
+*****************************************
+*   Opens for Input Only               *
+*****************************************
+         ORG   LOGREVENTDATA
+LOGROPENINPUT    EQU 3       Opens for input only event type
+LOGR#OICLASS     EQU 1       File class of file opened
+LOGR$OICLASS     DS  0CL8
+LOGR#OIDEVCLASS  EQU 2       Device class (from UCBCLASS)
+LOGR$OIDEVCLASS  DS  0XL1
+LOGR#OIDEVICE    EQU 3       Device name
+LOGR$OIDEVICE    DS  0CL8
+LOGR#OIOWNER     EQU 4       User ID of file owner
+LOGR$OIOWNER     DS  0CL8
+LOGR#OIFILE      EQU 5       File name of file opened
+LOGR$OIFILE      DS  0CL8
+LOGR#OILIB       EQU 6       Library of file opened
+LOGR$OILIB       DS  0CL8
+LOGR#OIVOL       EQU 7       Volume of file opened
+LOGR$OIVOL       DS  0CL8
+LOGR#OITYPE      EQU 8       Open type (from UFBF2)
+LOGR$OITYPE      DS  0XL1
+LOGR#OIERROR     EQU 9       Error on protection violation
+LOGR$OIERROR     DS  0XL1
+*****************************************
+*   Opens for Possible Modification    *
+*****************************************
+         ORG   LOGREVENTDATA
+LOGROPENMOD      EQU 4       Opens for possible modification event type
+LOGR#OMCLASS     EQU 1       File class of file opened
+LOGR$OMCLASS     DS  0CL8
+LOGR#OMDEVCLASS  EQU 2       Device class (from UCBCLASS)
+LOGR$OMDEVCLASS  DS  0XL1
+LOGR#OMDEVICE    EQU 3       Device name
+LOGR$OMDEVICE    DS  0CL8
+LOGR#OMOWNER     EQU 4       Userid of file owner
+LOGR$OMOWNER     DS  0CL8
+LOGR#OMFILE      EQU 5       Filename of file opened
+LOGR$OMFILE      DS  0CL8
+LOGR#OMLIB       EQU 6       Library of file opened
+LOGR$OMLIB       DS  0CL8
+LOGR#OMVOL       EQU 7       Volume of file Opened
+LOGR$OMVOL       DS  0CL8
+LOGR#OMTYPE      EQU 8       Open type (from UFBF2)
+LOGR$OMTYPE      DS  0XL1
+LOGR#OMERROR     EQU 9       Error on protection violation
+LOGR$OMERROR     DS  0XL1
```

```
+*************************************
+*    Close                          *
+*************************************
+          ORG   LOGREVENTDATA
+LOGRCLOSE       EQU 5           Close event type
+LOGR#CFILE      EQU 1           Filename of file closed
+LOGR$CFILE      DS  0CL8
+LOGR#CLIB       EQU 2           Library of file closed
+LOGR$CLIB       DS  0CL8
+LOGR#CVOL       EQU 3           Volume of file closed
+LOGR$CVOL       DS  0CL8
+LOGR#CDEVCLASS  EQU 4           Device class (from UCBCLASS)
+LOGR$CDEVCLASS  DS  0XL1
+LOGR#CDEVICE    EQU 5           Device name
+LOGR$CDEVICE    DS  0CL8
+LOGR#COPENTYPE  EQU 6           Open type (from UFBF2)
+LOGR$COPENTYPE  DS  0XL1
+*************************************
+*    Rename                         *
+*************************************
+          ORG   LOGREVENTDATA
+LOGRRENAME      EQU 6           Rename event type
+LOGR#RCLASS     EQU 1           File class of file renamed
+LOGR$RCLASS     DS  0CL8
+LOGR#ROWNER     EQU 2           User ID of file owner
+LOGR$ROWNER     DS  0CL8
+LOGR#ROFILE     EQU 3           Filename of old file
+LOGR$ROFILE     DS  0CL8
+LOGR#ROLIB      EQU 4           Library of old file
+LOGR$ROLIB      DS  0CL8
+LOGR#ROVOL      EQU 5           Volume of old file
+LOGR$ROVOL      DS  0CL8
+LOGR#RNFILE     EQU 6           File name of new file
+LOGR$RNFILE     DS  0CL8
+LOGR#RNLIB      EQU 7           Library of new file
+LOGR$RNLIB      DS  0CL8
+LOGR#RNVOL      EQU 8           Volume of new file
+LOGR$RNVOL      DS  0CL8
+LOGR#RTYPE      EQU 9           Type of rename
+LOGR$RTYPE      DS  0XL1
+LOGR@RTFILE     EQU 1           Rename of a file
+LOGR@RTLIB      EQU 2           Rename of a library
```

```
+**************************************
+*    Scratch                         *
+**************************************
+          ORG   LOGREVENTDATA
+LOGRSCRATCH      EQU 7          Scratch event type
+LOGR#SCLASS      EQU 1          File class of file to be scratched
+LOGR$SCLASS      DS  0CL8
+LOGR#SOWNER      EQU 2          User ID of file owner
+LOGR$SOWNER      DS  0CL8
+LOGR#SFILE       EQU 3          File name of file scratched
+LOGR$SFILE       DS  0CL8
+LOGR#SLIB        EQU 4          Library of file scratched
+LOGR$SLIB        DS  0CL8
+LOGR#SVOL        EQU 5          Volume of file scratched
+LOGR$SVOL        DS  0CL8

+**************************************
+*    Change File Attributes          *
+**************************************
+          ORG   LOGREVENTDATA
+LOGRCHNGFATTR    EQU 8          Change file attributes event type
+LOGR#CFACLASS    EQU 1          File class of file
+LOGR$CFACLASS    DS  0CL8
+LOGR#CFAOWNER    EQU 2          User ID of file owner
+LOGR$CFAOWNER    DS  0CL8
+LOGR#CFAFILE     EQU 3          File name of fiel scratched
+LOGR$CFAFILE     DS  0CL8
+LOGR#CFALIB      EQU 4          Library of file scratched
+LOGR$CFALIB      DS  0CL8
+LOGR#CFAVOL      EQU 5          Volume of file scratched
+LOGR$CFAVOL      DS  0CL8
+LOGR#CFAATTR     EQU 6          Attribute name
+LOGR$CFAATTR     DS  0CL16
+LOGR#CFAOLDVAL   EQU 7          Old attribute value
+LOGR$CFAOLDVAL   DS  0XL32        (data type depends on attribute)
+LOGR#CFANEWVAL   EQU 8          New attribute value
+LOGR$CFANEWVAL   DS  0XL32

+**************************************
+*    Security Program Usage          *
+**************************************
+          ORG   LOGREVENTDATA
+LOGRSECURITY     EQU 9          Security program usage event type
+*    ***************************************************
+*    *  Subtype Add User for Security Event Type       *
+*    ***************************************************
+LOGRSUBSECADD    EQU 1          Security subtype for add user
+LOGR#SUADUID     EQU 1          User ID of new user
+LOGR$SUADUID     DS  0CL8
+LOGR#SUADNAME    EQU 2          New username
+LOGR$SUADNAME    DS  0CL24
```

```
+*    ***************************************************
+*    *   Subtype Delete User for Security Event Type   *
+*    ***************************************************
+         ORG   LOGREVENTDATA
+LOGRSUBSUDEL     EQU 2           Security subtype for delete user
+LOGR#SUDUID      EQU 3           User ID deleted
+LOGR$SUDUID      DS   0CL8
+*    **************************************************************
+*    *   Subtype Change User Attributes for Security Event Type   *
+*    **************************************************************
+         ORG   LOGREVENTDATA
+LOGRSUBCUA       EQU 3           Security subtype for change user
+*                                attributes
+LOGR#SCUAUID     EQU 4           User ID whose attributes are being
+*                                changed
+LOGR$SCUAUID     DS   0CL8
+LOGR#SCUAATTR    EQU 5           Name of attribute being changed
+LOGR$SCUAATTR    DS   0CL16
+LOGR#SCUAOLD     EQU 6           Old attribute value
+LOGR$SCUAOLD     DS   0XL72
+LOGR#SCUANEW     EQU 7           New attribute value
+LOGR$SCUANEW     DS   0XL72

+*    *****************************************************************
+*    *   Subtype Change User Access Rights for Security Event Type   *
+*    *****************************************************************
+         ORG   LOGREVENTDATA
+LOGRSUBSUAC      EQU 4           Security subtype for change user access
+*                                rights

+LOGR#SUACUID     EQU 8           User ID whose access rights are being
+*                                changed
+LOGR$SUACUID     DS   0CL8
+LOGR#SUACCLASS   EQU 9           File class
+LOGR$SUACCLASS   DS   0CL8
+LOGR#SUACOLD     EQU 10          Access rights
+LOGR$SUACOLD     DS   0BL1
+LOGR@SUACEX      EQU X'80'       . Execute
+LOGR@SUACRD      EQU X'40'       . Read
+LOGR@SUACWR      EQU X'20'       . Write
+LOGR#SUACNEW     EQU 11          New access value
+LOGR$SUACNEW     DS   0BL1
```

```
+*    There are still more subtypes to define here.
+         ORG   LOGREVENTDATA
+LOGRSUBSPAC      EQU 5             Security subtype for change program
+*                                  access rights
+LOGR#SPAVOL      EQU 12            Program volume
+LOGR$SPAVOL      DS  0CL6
+LOGR#SPALIB      EQU 12            Program library
+LOGR$SPALIB      DS  0CL8
+LOGR#SPAFILE     EQU 12            Program file name
+LOGR$SPAFILE     DS  0CL8
+LOGR#SPACCLASS   EQU 9             File class for program access rights
+LOGR$SPACCLASS   DS  0CL8
+LOGR#SPACOLD     EQU 10            Access rights
+LOGR$SPACOLD     DS  0BL1
+LOGR@SPACEX      EQU X'80'         . Execute
+LOGR@SPACRD      EQU X'40'         . Read
+LOGR@SPACWR      EQU X'20'         . Write
+LOGR#SPACNEW     EQU 11            New access value
+LOGR$SPACNEW     DS  0BL1


+**************************************
+*    Program Invocations            *
+**************************************
+         ORG   LOGREVENTDATA
+LOGRPROGRAM      EQU 10            Program invocations event type
+LOGR#PRGCLASS    EQU 1             File class of program
+LOGR$PRGCLASS    DS  0CL8
+LOGR#PRGOWNER    EQU 2             Userid of owner of file
+LOGR$PRGOWNER    DS  0CL8
+LOGR#PRGFILE     EQU 3             File name of program
+LOGR$PRGFILE     DS  0CL8
+LOGR#PRGLIB      EQU 4             Library of program
+LOGR$PRGLIB      DS  0CL8
+LOGR#PRGVOL      EQU 5             Volume of program
+LOGR$PRGVOL      DS  0CL8


+**************************************
+*    Procedure Invocations          *
+**************************************
+         ORG   LOGREVENTDATA
+LOGRPROCEDURE    EQU 11            Procedure invocations event type
+LOGR#PROCCLASS   EQU 1             File class of procedure
+LOGR$PROCCLASS   DS  0CL8
+LOGR#PROCOWNER   EQU 2             Userid of owner of file
+LOGR$PROCOWNER   DS  0CL8
+LOGR#PROCFILE    EQU 3             File name of procedure
+LOGR$PROCFILE    DS  0CL8
+LOGR#PROCLIB     EQU 4             Library of procedure
+LOGR$PROCLIB     DS  0CL8
+LOGR#PROCVOL     EQU 5             Volume of procedure
+LOGR$PROCVOL     DS  0CL8
```

```
+***************************************
+*    Background Jobs                  *
+***************************************
+         ORG   LOGREVENTDATA
+LOGRBACKGRND    EQU 12          Background jobs event type
+*    Fields common to all subtypes within the background jobs
+*    event type.
+LOGR#BGSFILE    EQU 1           File name of procedure
+LOGR$BGSFILE    DS  0CL8
+LOGR#BGSLIB     EQU 2           Library of procedure
+LOGR$BGSLIB     DS  0CL8
+LOGR#BGSVOL     EQU 3           Volume of procedure

+LOGR$BGSVOL     DS  0CL8
+LOGR#BGSJOB     EQU 4           Job name used
+LOGR$BGSJOB     DS  0CL8
+*    ****************************************************
+*    *   Subtype Submit for Background Jobs Event Type *
+*    ****************************************************
+LOGRSUBBGSUB    EQU 1           Subtype submit background job
+LOGR#BGSCLASS   EQU 5           File class of procedure
+LOGR$BGSCLASS   DS  0CL8
+LOGR#BGSOWNER   EQU 6           User ID of file owner
+LOGR$BGSOWNER   DS  0CL8
+LOGR#BGSJOBCLAS EQU 7           Job class
+LOGR$BGSJOBCLAS DS  0CL1
+LOGR#BGSJOBTYPE EQU 8           Type of job
+LOGR$BGSJOBTYPE DS  0XL1
+LOGR@BGSPERM    EQU X'80'       . Permanent Job

+*    ************************************************************
+*    *   Subtype Job Initiation for Background Jobs Event Type *
+*    ************************************************************
+         ORG   LOGREVENTDATA
+LOGRSUBBGJINIT  EQU 2           Subtype job initiation
+*     There are no additional fields for this subtype
+*    *************************************************************
+*    *   Subtype Job Termination for Background Jobs Event Type *
+*    *************************************************************
+         ORG   LOGREVENTDATA
+LOGRSUBBGJTERM  EQU 3           Subtype job termination
+LOGR#BGJTWHY    EQU 9           Reason for job termination
+LOGR$BGJTWHY    DS  0XL1
+LOGR@BGJTNORM   EQU 1           . Normal completion
+LOGR@BGJTDIED   EQU 2           . Error completion (cancel condition,
+*                                 program exception, etc.)
+LOGR@BGJTTIME   EQU 3           . Expired time limit
+LOGR@BGJTOPER   EQU 4           . Cancelled by operator
```

```
+****************************************
+*    DP Print Jobs                     *
+****************************************
+         ORG   LOGREVENTDATA
+LOGRDPPRINT    EQU 13         DP print jobs event type
+*    Fields common to all subtypes within the print jobs
+*    event type.
+LOGR#DPSFILE   EQU 1          File name of print file
+LOGR$DPSFILE   DS  0CL8
+LOGR#DPSLIB    EQU 2          Library of print file
+LOGR$DPSLIB    DS  0CL8
+LOGR#DPSVOL    EQU 3          Volume of print file
+LOGR$DPSVOL    DS  0CL8

+*    ************************************************
+*    *   Subtype Submit for DP Print Jobs Event Type   *
+*    ************************************************
+LOGRSUBDPSUB   EQU 1          Subtype submit DP print job
+LOGR#DPSCLASS  EQU 5          File class of print file
+LOGR$DPSCLASS  DS  0CL8
+LOGR#DPSOWNER  EQU 6          User ID of file owner
+LOGR$DPSOWNER  DS  0CL8
+LOGR#DPSPRTCLAS EQU 7         Print class
+LOGR$DPSPRTCLAS DS  0CL1

+*    ************************************************************
+*    *   Subtype Job Initiation for DP Print Jobs Event Type    *
+*    ************************************************************
+         ORG   LOGREVENTDATA
+LOGRSUBDPJINIT EQU 2          Subtype job initiation
+LOGR#DPJIPRTR  EQU 8          Name of printer used
+LOGR$DPJIPRTR  DS  0CL8

+*    *************************************************************
+*    *   Subtype Job Termination for DP Print Jobs Event Type    *
+*    *************************************************************
+         ORG   LOGREVENTDATA
+LOGRSUBDPJTERM EQU 3          Subtype job termination
+LOGR#DPJTWHY   EQU 10         Reason for job termination
+LOGR$DPJTWHY   DS  0XL1
+LOGR@DPJTNORM  EQU 1          . Normal completion
+LOGR@DPJTREAD  EQU 2          . I/O errors reading input file
+LOGR@DPJTPRTR  EQU 3          . I/O errors on printer
+LOGR@DPJTOPER  EQU 4          . Cancelled by operator
```

```
+**************************************
+*    Mount                           *
+**************************************
+         ORG   LOGREVENTDATA
+LOGRMOUNT       EQU 15          Mount event type
+LOGR#MTVOLUME   EQU 1           Name of volume mounted
+LOGR$MTVOLUME   DS  0CL8
+LOGR#MTDEVCLAS  EQU 2           Device class of device
+LOGR$MTDEVCLAS  DS  0XL1
+LOGR#MTDEVICE   EQU 3           Device name
+LOGR$MTDEVICE   DS  0CL8
+LOGR#MTOWNER    EQU 4           User ID of owner of volume
+LOGR$MTOWNER    DS  0CL8
+LOGR#MTMOUNTER  EQU 5           User ID of mounter of volume
+LOGR$MTMOUNTER  DS  0CL8

+**************************************
+*    Dismount                        *
+**************************************
+         ORG   LOGREVENTDATA
+LOGRDISMOUNT    EQU 16          Dismount event type
+LOGR#DMVOLUME   EQU 1           Name of volume dismounted
+LOGR$DMVOLUME   DS  0CL8
+LOGR#DMDEVCLAS  EQU 2           Device class of device
+LOGR$DMDEVCLAS  DS  0XL1
+LOGR#DMDEVICE   EQU 3           Device name
+LOGR$DMDEVICE   DS  0CL8
+LOGR#DMOWNER    EQU 4           User ID of owner of volume
+LOGR$DMOWNER    DS  0CL8
+LOGR#DMMOUNTER  EQU 5           User ID of dismounter of volume
+LOGR$DMMOUNTER  DS  0CL8
+**************************************
+*    Operator - User Communications  *
+**************************************
+         ORG   LOGREVENTDATA
+LOGROPERUSER    EQU 17          Operator-user communications event type
+LOGR#OUSENDER   EQU 1           Sender of message
+LOGR$OUSENDER   DS  0XL1
+LOGR@OUSOPER    EQU 1           . Operator sent to user
+LOGR@OUSUSER    EQU 2           . User sent to operator
+LOGR#OUDEVFROM  EQU 2           Name of device message sent from
+LOGR$OUDEVFROM  DS  0CL8
+LOGR#OUDEVTO    EQU 3           Name of device message sent to
+LOGR$OUDEVTO    DS  0CL8
+LOGR#OUTOUSER   EQU 4           User ID message sent to
+LOGR$OUTOUSER   DS  0CL8
+LOGR#OUFRUSER   EQU 5           User ID message sent from
+LOGR$OUFRUSER   DS  0CL8
+LOGR#OUOPEROPT  EQU 6           Option used from operator screen
+LOGR$OUOPEROPT  DS  0XL1
```

```
+LOGR@OUOPTONE   EQU 1          . Send to single user
+LOGR@OUOPTONEI  EQU 2          . Send to single user immediately
+LOGR@OUOPTALL   EQU 3          . Send to all users
+LOGR@OUOPTALLI  EQU 4          . Send to all users immediately
+LOGR@OUOPTREM   EQU 5          . Remove current messages
+LOGR#OUMESSAGE  EQU 7          The message sent
+LOGR$OUMESSAGE  DS  0CL160

+**************************************
+*    System Messages to Operator     *
+**************************************
+         ORG   LOGREVENTDATA
+LOGRSYSOPER     EQU 18         System messages to operator event type
+LOGR#SOMESSAGE  EQU 1          The message sent
+LOGR$SOMESSAGE  DS  0CL160

+**************************************
+*    Attach/Detach                   *
+**************************************
+         ORG   LOGREVENTDATA
+LOGRATTDET      EQU 19         Attach/detach devices event type
+LOGR#ADWHICH    EQU 1          Attach or detach
+LOGR$ADWHICH    DS  0XL1
+LOGR@ADATTACH   EQU 1          . Attach
+LOGR@ADDETACH   EQU 2          . Detach
+LOGR#ADDEVCLAS  EQU 2          Device class of device
+LOGR$ADDEVCLAS  DS  0XL1
+LOGR#ADDEVICE   EQU 3          Name of device attached or detached
+LOGR$ADDEVICE   DS  0CL8
+LOGR#ADUSER     EQU 4          User ID of operator
+LOGR$ADUSER     DS  0CL8
+LOGR#ADOPRDEV   EQU 5          Name of device used as operator
+LOGR$ADOPRDEV   DS  0CL8

+**************************************
+*    Acquire/Release                 *
+**************************************
+         ORG   LOGREVENTDATA
+LOGRACQREL      EQU 20         Acquire/release devices event type
+LOGR#ARWHICH    EQU 1          Acquire or Release
+LOGR$ARWHICH    DS  0XL1
+LOGR@ARACQUIRE  EQU 1          . Acquire
+LOGR@ARRELEASE  EQU 2          . Release
+LOGR#ARDEVCLAS  EQU 2          Device class of device
+LOGR$ARDEVCLAS  DS  0XL1
+LOGR#ARDEVICE   EQU 3          Name of device acquired or released
+LOGR$ARDEVICE   DS  0CL8

+LOGR#ARUSER     EQU 4          User ID of operator
+LOGR$ARUSER     DS  0CL8
+LOGR#AROPRDEV   EQU 5          Name of device used as operator
+LOGR$AROPRDEV   DS  0CL8
```

```
+**************************************
+*    System Snapshots                *
+**************************************
+          ORG   LOGREVENTDATA
+LOGRSNAPS         EQU 21          System snapshots event type
+LOGR#SNUSER       EQU 1           User ID of operator
+LOGR$SNUSER       DS  0CL8
+LOGR#SNOPRDEV     EQU 2           Name of device used as operator
+LOGR$SNOPRDEV     DS  0CL8

+**************************************
+*    Logger Messages                 *
+**************************************
+          ORG   LOGREVENTDATA
+LOGRLOGGER        EQU 22          Logger messages event type
+LOGR#LMSGTYPE     EQU 1           Type of message
+LOGR$LMSGTYPE     DS  0XL1
+LOGR@LSTARTNEW    EQU 1         Logging started because of Newlog command
+LOGR@LCONT        EQU 2           Logging started because of Continue
+*                                 command
+LOGR@LSTOP        EQU 3           Logging stopped because Stop command
+*                                 issued
+LOGR@LSTOPNL      EQU 4           Logging stopped because Newlog command
+*                                 issued
+LOGR@LRSTEVNTS    EQU 5           Reset events issued                 01\
+LOGR@LSETEVNTS    EQU 6           Set events issued                   01\
+LOGR@LRSTVIOLS    EQU 7           Reset violations issued             01\
+LOGR@LSETVIOLS    EQU 8           Set violations issued               01\
+LOGR@LOLDBAD      EQU 9           New file opened because old one bad 02\
+LOGR@LIPL         EQU 10          Logging continued because of IPL    04\
+LOGR#LUSER        EQU 2           User ID of CNTROLOG caller
+LOGR$LUSER        DS  0CL8
+LOGR#LDEVICE      EQU 3           Name of device used
+LOGR$LDEVICE      DS  0CL8
+LOGR#LOLDMAP      EQU 4           Previous event or violations bit map01\
+*                                  (before set or reset issued)       01\
+LOGR$LOLDMAP      DS  0BL.256                                         01\
+LOGR#LNEWMAP      EQU 5           New event or violations bit map     01\
+*                                 (as a result of set or reset issued)01\
+LOGR$LNEWMAP      DS  0BL.256                                         01\
+**************************************
+* Operator message to Logging Task   *
+**************************************
+         ORG   LOGREVENTDATA
+LOGROPRMSG      EQU 23          Operator message to logging task event
+*                               Type
+LOGR#OMUSER     EQU 1           User ID of operator
+LOGR$OMUSER     DS  0CL8
+LOGR#OMOPRDEV   EQU 2           Name of device used as operator
+LOGR$OMOPRDEV   DS  0CL8
+LOGR#OMMESSAGE  EQU 3           The text of the message sent
+LOGR$OMMESSAGE  DS  0CL160
+         ORG ,
+LOGRLENGTH     EQU   *-LOGRBEGIN
+                CSECT
          END BEGIN
```

3.2.3 MSMAP - Map Region of Virtual Address Space

Syntax

```
[label]  MSMAP       RETURNCODE=returncode,
                     PATHNAME=pathname,
                     TYPE=type,
                   [ OPTION=option,]
                     COMMAND=command,
                   [,READLEVEL=readlevel]
                   [,WRITELEVEL=writelevel]
                    ,STRTADDR=strtaddr
                   [,LOWERVA=lowerva,]
                   [ UPPERVA=upperva]
                   [,FILESIZE=filesize]
                   [,FILECLS=fileclass]
```

Function

MSMAP provides for mapping program and data files into a task's virtual address space. The file may already exist or may be opened in exclusive or shared mode when issuing the MSMAP call. Through the COMMAND parameter, the user may specify mapping at a specific address, within a range of addresses or at any available address. The recommended choice is at any available address. MSMAP returns the mapped address through the STRTADDR parameter. Files must be aligned on a 1/8-MB boundary.

The caller may also specify the process level required to read or write to the file. For all nonprivileged code, the read and write levels default to process level 0.

Parameter Definitions

Parameter Definition	I/O	Data Type
command	Input	fixed bin(15,0)

Specifies the address at which the file is to be mapped. The values may be 0, 1, or 2. A value of 0 means to map the file at any available location and is the recommended choice. A value of 1 means to map at any available location between the addresses specified by the LOWERVA and UPPERVA parameters. A value of 2 means to map at the address specified by the STRTADDR parameter. In all three cases, the file must be aligned on a 1/8-MB boundary. If COMMAND=2, this means that STRTADDR must specify an address which is an integer multiple of 1/8-MB. If COMMAND=1, then the range specified by LOWERVA and UPPERVA must contain at least one 1/8-MB boundary.

fileclass Input char(1)

FILECLS specifies the file security access code of the data file to be mapped (such as '#' or 'A' or ' '). If FILECLS is not specified, the default output file class of the caller is used for the mapped data file. This parameter is only used when creating a data file.

filesize Input fixed bin(31,0)

FILESIZE specifies the initial size (in bytes) of the file. This parameter is required only when creating a new data file.

lowerva Input pointer

Specifies the lower virtual address limit at which the file may be mapped. Required only if specifying a range of addresses for the map (COMMAND=1). UPPERVA must also be supplied.

option Input fixed bin(15,0)

OPTION specifies that the data file already exists or is to be created. This parameter is required only if mapping an exclusive data file (TYPE=2) or a shared data file (TYPE=3). A value of 0 means to map an existing data file with a specified name. A value of 2 means to create and map a data file with a specified name and file size.

pathname Input char(22) varying

PATHNAME specifies the volume, library, and file name of the file to be mapped. The parameter must be generated as follows: a 6-byte volume name plus an eight byte library name and an 8-byte file name. If the actual values for the volume, library and file consist of fewer characters than the allocated size, they must be left-justified and padded with blanks.

readlevel Input fixed bin(15,0)

Contains the read level of the region to be mapped. READLEVEL defaults to 0 if it is not specified and is the only valid level for nonprivileged code.

returncode Output fixed bin(31,0)

Code that indicates the success or failure of the routine call.

strtaddr I/O pointer

As an input parameter, STRTADDR specifies a specific virtual address at which to map the file. As an output parameter, STRTADDR contains the actual virtual address that was mapped.

type Input fixed bin(15,0)

The TYPE parameter specifies whether the file is a program file, an exclusive data file or a shared data file. A value of 1 means to map a program file. A value of 2 means to map an exclusive data file. A value of 3 means to map a shared data file.

upperva Input pointer

Contains the upper limit of the highest virtual address of the region to be mapped. Required only if mapping within a range of addresses (COMMAND=1).

writelevel Input fixed bin(15,0)

Contains the write level of the region to be mapped. WRITELEVEL defaults to 0 if it is not specified and is the only valid level for nonprivileged code.

Return Codes

The return codes listed here are the base values. Some values may be returned that are 1000, 2000, 3000, 4000, or 5000 plus the base values. However, the definition of the return code does not change. For example, the return code values of 16 and 1016 have the same definitions.

Code	Definition
@ERSUCC	Success
@ERIPVAL	Illegal parameter value
@ERPROT	Attempted protection violation
4	File already mapped with different conditions
8	File not found or inaccessible
12	Library not found or inaccessible
16	Volume not mounted or inaccessible
20	All buffers in use
24	VTOC errors
28	I/O error on VTOC
32	Buffer size not enough for all FDRs
36	READFDR failed

Code	Definition
40	Not enough virtual space to map the file
44	GETMEM failure during region node creation
48	Region node table full (max. # of nodes exist)
60	Caller not privileged enough
64	Bad parameter list
88	Access rights denied
92	No space on volume or VTOC
96	CREATFDR failed
100	Bad object format
104	Stack space not available for static allocation
108	Insufficient data area space.
110	Error from MAP called by MAP.
112	Unresolved SSL references.
116	Too many SSL files.
120	Could not open alias file.
200	Subroutine not set up for specified file type - invalid mode.

Example

```
          MSMAP RETURNCODE=RCODE,PATHNAME=FSPEC,TYPE=DATAFILE
                COMMAND=ANY,STRTADDR=HERE
+         DS    0H
+         PUSHA 0,0
+         OI    0(SP),X'80'        flag last argument
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,DATAFILE
+         PUSHA 0,FSPEC
+         PUSHA 0,RCODE
+#MSMAP   STATIC
+         ORG   #MSMAP
+         DC    V(MSMAP)
+         CSECT
+         L     R1,=R(#MSMAP)
+         L     R1,0(R14,R1)
+         PUSH  0,R1
+         LA    R1,4(,SP)
+         JSI   0(,SP)
+         POPN  0,12*4+4

                 .
          (Static Section)
                 .

 RCODE    DS F
 FSPEC    DC CL22'TDATA   MYLIBRY SYSTEM'
 DATAFILE DC H'0'
 ANY      DC H'0'
 HERE     DS A
```

3.2.4 MSUNMAP - Unmap Region of Virutal Address Space

Syntax

```
[label]  MSUNMAP   RETURNCODE=returncode,
                   PATHNAME=pathname
```

Function

Unmaps a file from a task's virtual address space.

Parameter Definitions

Parameter Definition	I/O	Data Type
pathname	Input	char(22) varying

Contains the volume, library and file name of the data file to be unmapped. The parameter must be generated as follows: a 6-byte volume name plus an 8-byte library name and an 8-byte file name. If the actual values for the volume, library and file consist of fewer characters than the allocated size, they must be left-justified and padded with blanks.

returncode	Output	fixed bin(31,0)

Contains a code that indicates the success or failure of the routine call.

Return Codes

Code	Definition
0	Success
4	File not mapped on present link level
8	Caller not privileged enough
12	Error from FREEHEAP

Example

```
 DOMAP     MSUNMAP RETURNCODE=RCODE,PATHNAME=FSPEC
+DOMAP     EQU    *
+          PUSHA 0,FSPEC
+          OI     0(SP),X'80'         flag last argument
+          PUSHA 0,RCODE
+#MSUNMAP STATIC
+          ORG    #MSUNMAP
+          DC     V(MSUNMAP)
+          CSECT
+          L      R1,=R(#MSUNMAP)
+          L      R1,0(R14,R1)
+          PUSH   0,R1
+          LA     R1,4(,SP)
+          JSI    0(,SP)
+          POPN   0,2*4+4

                  .
          (Static Section)
                  .


 RCODE     DS F
 FSPEC     DC CL22'TDATA   MYLIBRY SYSTEM'
           END BEGIN
```

3.2.5 PROCINFO - Process Information

Syntax

```
[label]  PROCINFO    RETURNCODE=returncode
                    [,PROCESSID=processid]
                    [,PARENTID=parentid]
                    [,MYPID=mypid]
                    [,WSNUM=wsnum]
                    [,DEBUGSTATUS=debugstatus]
```

Function

PROCINFO provides user programs with information related to a specific process or task. Information is provided for the caller's process/task or other processes/tasks if the correct ID is known.

Parameter Definitions

Parameter Definition	I/O	Data Type
debugstatus	Output	fixed bin(15,0)

A nonzero value indicates that the process or task specified by PROCESSID is under control of the system debugger program.

mypid	Output	fixed bin(31,0)

The process ID or task number of the caller.

parentid	Output	fixed bin(31,0)

Contains the process ID or task number of the parent task, if any. If there is no parent task, this value is 0.

processid	Input	fixed bin(31,0)

The process or task ID of this request.

returncode	Output	fixed bin(31,0)

A code that indicates the success or failure of the routine call.

wsnum		Output fixed bin(31,0)

Contains the workstation number of the process or task specified by PROCESSID.

Return Codes

Code	Definition
@ERSUCC	Success
@ERBADPID	Process ID specified and not found

Example

```
 GETINFO   PROCINFO RETURNCODE=RCODE,MYPID=MYTASK,WSNUM=MYTUBE
+GETINFO   DS    0H
+          PUSHA 0,0
+          OI    0(SP),X'80'        flag last argument
+          PUSHA 0,MYTUBE
+          PUSHA 0,MYTASK
+          PUSHA 0,0
+          PUSHA 0,0
+          PUSHA 0,RCODE
+#PROCINF  STATIC
+          ORG   #PROCINF
+          DC    V(PROCINFO)
+          CSECT
+          L     R1,=R(#PROCINF)
+          L     R1,0(R14,R1)
+          PUSH  0,R1
+          LA    R1,4(,SP)
+          JSI   0(,SP)
+          POPN  0,6*4+4

                     .
           (Static Section)
                     .

 RCODE     DS F
 MYTASK    DS F
 MYTUBE    DS F


 GETINFO   PROCINFO RETURNCODE=RCODE,PROCESSID=PID,PARENTID=DAD,          -
                 DEBUGSTATUS=INDEBUG,WSNUM=TUBENUM
+GETINFO   DS    0H
+          PUSHA 0,INDEBUG
+          OI    0(SP),X'80'        flag last argument
+          PUSHA 0,TUBENUM
+          PUSHA 0,0
+          PUSHA 0,DAD
+          PUSHA 0,PID
+          PUSHA 0,RCODE
```

```
+#PROCINF STATIC
+         ORG   #PROCINF
+         DC    V(PROCINFO)
+         CSECT
+         L     R1,=R(#PROCINF)
+         L     R1,0(R14,R1)
+         PUSH  0,R1
+         LA    R1,4(,SP)
+         JSI   0(,SP)
+         POPN  0,6*4+4

                 .
          (Static Section)
                 .


 RCODE    DS F
 PID      DS F
 DAD      DS F
 INDEBUG  DS H
```

3.2.6 PUTLOG - Put Record Into System Security Database File

Syntax

```
[label] PUTLOG       RC=returncode,
                    [,TYPE=type]
                    [,SUBTYPE=subtype]
                    [,VIOLATION=violation]
                    [,WAIT=wait]
                    [,DATA=data]
                    [,SUBJUID=userid]
                    [,SUBJWS=workstation]
                    [,SUBJJOB=jobname]
                    [,SUBJTASKID=subjtaskid]
```

Function

PUTLOG inserts a record into the system security event logging database file. Two hundred, fifty-six characters of event-related user data may be stored in the record along with the job name, task ID, user ID, workstation, and event type.

Parameter Definitions

Parameter Definition	I/O	Data Type
data	Input	char(256) var

The information to be logged. Up to 256 characters of information may be recorded.

jobname	Input	char(8) var

The job name used by the subject of the PUTLOG message.

returncode	Output	fixed bin(31,0)

A code that specifies the success or failure of the routine call.

subtaskid	Input	fixed bin(15,0)

The task ID of the subject of the PUTLOG.

subtype	Input	fixed bin(15,0)

An integer that enables finer distinctions, within TYPE, for the information being logged. This is for informational purposes only and can be defined by the user. SUBTYPE defaults to 0.

type Input fixed bin(15,0)

An integer that corresponds to the type of event to be logged. Each TYPE corresponds directly to a bit in the events bit mask modifiable via the CNTROLOG System Service. The TYPE specified is validated against the privilege level of the caller. The default is 0.

userid Input char(8) var

The user ID of the subject of the PUTLOG message.

violation Input fixed bin(15,0)

If the specified value is 1, PUTLOG marks the event as an attempted violation. The default is 0.

wait Input fixed bin(15,0)

If the value specified is 1, PUTLOG will wait for the IPC message to be sent to the logging task. If 0, a NOWAITSEND will be specified on the IPC-generated ISEND and PUTLOG will not wait for the message to be sent. The default is 1.

workstation Input char(8) var

The workstation being used by the subject of the PUTLOG message.

<u>Return Codes</u>

Code	Definition
@ERLOGEVNTNOTSET	Event specified on PUTLOG is not set to be logged.
@ERLOGINACTIVE	Logging is not active.
@ERLOGNOTPRIV	Caller not authorized to log this event type.
@ERLOGVIOLNOTSET	Violation specified on PUTLOG is not set to be logged.
@ERNOLOGGING	Logging task has been terminated.

Example

```
 LOGR#USER EQU 0
          PUTLOG RC=RCODE,TYPE=EVTTYPE,DATA=DATAMSG
+         PUSHA 0,DATAMSG              . Data to be Logged .
+         OI    0(15),X'80'            . Indicate Parameter List End .
+         PUSHA 0,0                    . (Wait) .
+         PUSHA 0,0                    . (VIOLATION Parameter) .
+         PUSHA 0,0                    . (Subtype) .
+         PUSHA 0,EVTTYPE              . Type .
+         PUSHA 0,RCODE                . Return Code .
+#PUTLOG  STATIC                       . Section for PUTLOG VCON .
+         ORG   #PUTLOG                . Start the section ... .
+         DC    V(PUTLOG)              . ... with the VCON .
+         CSECT                        . Rejoin current section .
+         L     1,=R(#PUTLOG)          . Address Static Section .
+         L     1,0(14,1)              . Add Static Base .
+         PUSH  0,1                    . Enstack VCON Address .
+         LA    1,4(,15)               . Address Parameters .
+         JSI   0(,15)                 . Call PUTLOG .
+         POPN  0,28                   . Restore Stack .

                .
          (Static Section)
                .

 EVTTYPE  DC Y(LOGR#USER)
 RCODE    DC F'0'
 DATAMSG  DC H'3'
          DC CL3'EVT'
```

3.2.7 SBREAK - Break Synchronization

```
[label]  SBREAK      RETURNCODE=returncode,
                   [ NUMHAN=numhan,]
                     HANDLES=handles
                   [,CANCEL={YES}]
                           {NO }
```

Function

Break synchronization allow users to handle a task that has hung while holding a synchronous object. It will remove the task that is holding the object and gives the object to the task which issued the break synchronization call. If the requesting task is the current user of the synchronous object then the calling task remains the user and the cancel option (if specified) is ignored. If there is no user of the object then the caller is given control of the object and the cancel option (if specified) is ignored. This service therefore allows users to do cleanup if necessary before further damage is done (particularly with partial data base updates).

If the synchronous object was created with the RESTRICT option, the break synchronization call can only be accepted from the object creator. Any other callers are not accepted. If the RESTRICT option was not specified at create time, any task can do a break.

The CANCEL parameter allows the break caller to specify that the task which previously held the synchronous object is to be cancelled. The task being cancelled has any CEXITS disabled. If there is a fatal task crash, the cancel is ineffective. Some other cases may also cause the task not to cancel. However, in no known case does task remain in user code.

Parameter Definitions

Parameter Definition	I/O	Data Type
cancel	Input	fixed bin(31,0)

CANCEL specifies whether the task currently holding the synchronous object should be cancelled. A value of 1 indicates that it should be cancelled. A value of 0 indicates that it should not.

Parameter Definition	I/O	Data Type
handles	Input	pointer to an array of char (8) entries

HANDLES specifies a pointer to an array of 8-character handle identifiers. The array is currently restricted to one entry.

numhan Input fixed bin(31,0)

NUMHAN indicates how many handles there are in the call. Defaults to 1. Currently restricted to one handle.

returncode Output fixed bin(31,0)

Code that indicates the success or failure of the routine call.

Return Codes

Code	Definition
@ERSUCC	Success
@ERIPVAL	Invalid parameter
@ERNOTFOUND	Synchronous object not found
@ERNOTOWN	Access denied
@ERSOACCDIS	Access denied (call from anynchronous exit)
@ERPROT	Protection violation
@ERMISALIGN	Parameter misaligned
@ERIPTYP	Invalid parameter type
@ERUNPRIV	Caller does not have high enough process level for this object

Example

```
          SBREAK RETURNCODE=RTC,NUMHAN=NUM,HANDLES=HANPTR,                 -
                CANCEL=YES
+         PUSHA 0,=A(1)             INDICATE CANCEL REQUESTED
+         OI    0(SP),X'80'         FLAG LAST ARGUMENT
+         PUSHA 0,HANPTR            Handle pointer
+         DS    0H
+         PUSHA 0,NUM               Number of handles in array
+         PUSHA 0,RTC               Return code
```

```
+#SBREAK  STATIC
+         ORG    #SBREAK
+         DC     V(SBREAK)
+         CSECT
+         L      R1,=R(#SBREAK)       Address of routine to call
+         L      R1,0(R14,R1)
+         PUSH   0,R1
+         LA     R1,4(,SP)            Point to argument list
+         JSI    0(,SP)               Call SBREAK
+         POPN   0,4*4+4

                  .
          (Static Section)
                  .


 RTC      DS F
 NUM      DC F'1'
 HANPTR   DC A(HANDLES)
 HANDLES  DS D
```

3.2.8 SCREATE - Create Synchronization Object

```
[label]  SCREATE     RETURNCODE=returncode,
                   [ NUMHAN=numhan,]
                     HANDLES=handles
                   [,RESTRICT={YES}]
                              {NO }
```

Function

Create synchronization object creates a data structure that controls the use of a shared resource. A synchronization object must be created before it can be used.

Each time SCREATE is called, a new synchronization object is created for the caller with a unique identifier (handle). The synchronization object creator then passes the handle to any other users of this synchronization object.

Parameter Definitions

Parameter Definition	I/O	Data Type
handles	Output	pointer to an array of char (8) entries

HANDLES specifies a pointer to an array of 8-character identifiers. The parameter is currently restricted to one entry in the array.

numhan	Input	fixed bin(31,0)

NUMHAN indicates how many handles are specified in the HANDLES array. The default is 1 and the parameter is currently restricted to one handle.

restrict	Input	fixed bin(31,0)

The RESTRICT parameter allows the creator to impose some restrictions on a synchronization object. If RESTRICT is set to yes (a value of 1), only the creator can issue a delete or break on the synchronization object. If RESTRICT is set to no (a value of 0), or omitted, any caller may issue a delete or break.

returncode	Output	fixed bin(31,0)

Code that indicates the success or failure of the routine call.

Return Codes

Code	Definition
@ERSUCC	Success
@ERIPVAL	Invalid parameter
@ERPROT	Protection violation
@ERMISALIGN	Parameter misaligned
@ERIPTYP	Invalid parameter type
@ERSOACCDIS	Access disallowed (call from asynchronous exit which is not allowed)

Example

```
          SCREATE RETURNCODE=RTC,NUMHAN=NUM,HANDLES=HANPTR,              -
                RESTRICT=NO
+         DS    0H
+         PUSHA 0,=A(0)          No restriction
+         OI    0(SP),X'80'      Flag last argument
+         PUSHA 0,HANPTR         Handle pointer
+         DS    0H
+         PUSHA 0,NUM            Number of handles in array
+         PUSHA 0,RTC            Return code
+#SCREATE STATIC
+         ORG   #SCREATE
+         DC    V(SCREATE)
+         CSECT
+         L     R1,=R(#SCREATE)  Address of routine to call
+         L     R1,0(R14,R1)
+         PUSH  0,R1
+         LA    R1,4(,SP)        Point to argument list
+         JSI   0(,SP)           Call SCREATE
+         POPN  0,4*4+4

                 .
          (Static Section)
                 .

 RTC      DS F
 NUM      DC F'1'
 HANPTR   DC A(HANDLES)
 HANDLES  DS D
```

3.2.9 SDELETE - Delete Synchronization Object

Syntax

```
[label]  SDELETE    RETURNCODE=returncode,
                   [NUMHAN=numhand,]
                    HANDLES=handles
```

Function

Delete synchronization object marks a synchronous object for delete, thereby disallowing any new waiters to enter the queue. The caller must first issue a call to SENTER to be able to delete the object. If there are no tasks currently waiting on the object, the synchronous object is deleted promptly. Any tasks that are waiting at the time of the delete can proceed normally. In this case, the object is deleted when all waiting tasks have been serviced.

If the object was created with the RESTRICT option, only the creator can successfully issue the delete call. Otherwise, any caller can issue the delete.

Parameter Definitions

Parameter Definition	I/O	Data Type
handles	Input	pointer to an array of char (8) entries

HANDLES specifies a pointer to an array of 8-character identifiers. The number of identifiers in the array is specified with the NUMHAN parameter. Currently there is a restriction of one identifier per routine call. The default is 1.

Parameter Definition	I/O	Data Type
numhan	Input	fixed bin(31,0)

The NUMHAN parameter indicates how many handles are to be created in this call. The default is 1 and is currently restricted to one.

Parameter Definition	I/O	Data Type
returncode	Output	fixed bin(31,0)

Code that indicates success or failure of the call.

Return Codes

Code	Definition
@ERSUCC	Success
@ERIPVAL	Invalid parameter
@ERNOTFOUND	Synchronous object not found
@ERNOTOWN	Access disallowed
@ERSOACCDIS	Access disallowed (call from anynchronous exit)
@ERPROT	Protection violation
@ERMISALIGN	Parameter misaligned
@ERIPTYP	Invalid parameter type
@ERUNPRIV	Caller does not have high enough process level for this object
@ERMKDEL	Object marked for delete (will be deleted when last of current waiters has finished)

Example

```
          SDELETE RETURNCODE=RTC,NUMHAN=NUM,HANDLES=HANPTR,
+         PUSHA 0,HANPTR            Handle pointer
+         OI    0(SP),X'80'         Flag last argument
+         DS    0H
+         PUSHA 0,NUM               Number of handles in array
+         PUSHA 0,RTC               Return code
+#SDELETE STATIC
+         ORG   #SDELETE
+         DC    V(SDELETE)
+         CSECT
+         L     R1,=R(#SDELETE)     Address of routine to call
+         L     R1,0(R14,R1)
+         PUSH  0,R1
+         LA    R1,4(,SP)           Point to argument list
+         JSI   0(,SP)              Call SDELETE
+         POPN  0,3*4+4

                .
          (Static Section)
                .
```

```
RTC      DS F
NUM      DC F'1'
HANPTR   DC A(HANDLES)
HANDLES  DS D
```

3.2.10 SENTER - Enter Synchronization

Syntax

```
[label]  SENTER       RETURNCODE=returncode,
                    [ NUMHAN=numhan,]
                      HANDLES=handles
                    [,NOWAIT={YES}]
                             {NO }
```

Function

Issues the request to gain control of the synchronization object in order to use the resource. If no other user has control of the synchronization object, the caller receives control of the synchronization object. Control is then be passed back to the caller who may then proceed to use the resource.

If some other user is holding the synchronization object when a caller requests it, the caller is blocked and has to wait for the resource on a first in/first out queue. When the resource becomes available, control is returned to the next caller in the queue, who can then use the resource.

The NOWAIT parameter allows users not to block if the resource is not free, but to return to the caller with a return code indicating that the resource is not free. This allows callers to process other work while waiting for the resource to become free.

In some cases, users can receive error return codes from enter synchronization (see below). Therefore, callers must check the return code before assuming that they have control of the synchronization object.

Parameter Definitions

Parameter Definition	I/O	Data Type
handles	Input	pointer to an array of char (8) entries

HANDLES specifies a pointer to an array of 8-character identifiers. Currently restricted to one entry in the array.

nowait	Input	

Specifying YES indicates not to wait if a resource is not available. Specifying NO indicates to wait for resource availability.

numhan	Input	fixed bin(31,0)

Indicates how many handles in the call. The default is one and is a current system restriction.

returncode   Output   fixed bin(31,0)

Code that indicates the success or failure of the routine call.

Return Codes

Code	Definition
@ERSUCC	Success
@ERIPVAL	Invalid parameter
@ERNOTFOUND	Synchronous object not found
@ERSOUNAV	Synchronous object unavailable (for NOWAIT option)
@ERSOACCDIS	Access to synchronous object disallowed (for async exits)
@ERALRDYHAS	User already has control of synchronous object
@ERPROT	Protection violation
@ERMISALIGN	Parameter misaligned
@ERIPTYP	Invalid parameter type
@ERUNPRIV	Caller does not have the correct process level for this object

Example

```
         SENTER RETURNCODE=RTC,NUMHAN=NUM,HANDLES=HANPTR,                    -
               NOWAIT=NO
+        DS     0H
+        PUSHA 0,=A(0)            No NOWAIT
+        OI     0(SP),X'80'       Flag last argument
+        PUSHA 0,HANPTR           Handle pointer
+        DS     0H
+        PUSHA 0,NUM              Number of handles in array
+        PUSHA 0,RTC              Return code
```

```
+#SENTER  STATIC
+         ORG   #SENTER
+         DC    V(SENTER)
+         CSECT
+         L     R1,=R(#SENTER)       Address of routine to call
+         L     R1,0(R14,R1)
+         PUSH  0,R1
+         LA    R1,4(,SP)            Point to argument list
+         JSI   0(,SP)               Call SENTER
+         POPN  0,4*4+4

                 .
          (Static Section)
                 .

 RTC      DS F
 NUM      DC F'1'
 HANPTR   DC A(HANDLES)
 HANDLES  DS D
```

3.2.11 SEXIT - Exit Synchronization

Syntax

```
[label]  SEXIT      RETURNCODE=returncode,
                   [NUMHAN=numhan,]
                    HANDLES=handles
```

Function

Exit synchronization releases the caller from control of the resource, and activates the first waiter.

Parameter Definitions

Parameter Definition	I/O	Data Type
handles	Input	pointer to an array of char (8) entries

HANDLES specifies a pointer to an array of handle identifiers. Currently restricted to one entry in the array.

numhan	Input	fixed bin(31,0)

Indicates how many handles in the call. The default is 1 and is the current system restriction.

returncode	Output	fixed bin(31,0)

Code that indicates the success or failure of the routine call.

Return Codes

Code	Definition
@ERSUCC	Success
@ERIPVAL	Invalid parameter
@ERNOTFOUND	Synchronous object not found
@ERNOTOWN	Synchronous object not owned by caller
@ERPROT	Protection violation
@ERMISALIGN	Parameter misaligned
@ERIPTYP	Invalid parameter type

Code	Definition
@ERUNPRIV	Caller does not have the correct process level for this object
@ERSOACCDIS	Access to synchronous object disallowed (for async exits)

Example

```
          SEXIT RETURNCODE=RTC,NUMHAN=NUM,HANDLES=HANPTR
+         PUSHA 0,HANPTR              Handle pointer
+         OI    0(SP),X'80'           Flag last argument
+         DS    0H
+         PUSHA 0,NUM                 Number of handles in array
+         PUSHA 0,RTC                 Return code
+#SEXIT   STATIC
+         ORG   #SEXIT
+         DC    V(SEXIT)
+         CSECT
+         L     R1,=R(#SEXIT)         Address of routine to call
+         L     R1,0(R14,R1)
+         PUSH  0,R1
+         LA    R1,4(,SP)             Point to argument list
+         JSI   0(,SP)                Call SEXIT
+         POPN  0,3*4+4

                .
          (Static Section)
                .

 RTC      DS F
 NUM      DC F'1'
 HANPTR   DC A(HANDLES)
 HANDLES  DS D
```

3.2.12 TCOMPLET - Check Task for Completion

Syntax

```
[label]  TCOMPLET  RETCODE=retcode,
                   TASKID=taskid
```

Function

This service allows a parent task to check on the completion of its child task. TCOMPLET does not return control to the calling task until the child task and its descendants have finished executing. When completed, all resources are released and TCOMPLET returns to the caller. A parent task should call either TKILL or TCOMPLET for all subtasks before its own completion.

Parameter Definitions

Parameter Definition	I/O	Data Type
retcode	Output	fixed bin(31,0)

Code that indicates the success or failure of the routine call.

Parameter Definition	I/O	Data Type
taskid	Input	fixed bin(31,0)

Specifies the task number of the subtask to be logged off and cancelled.

Return Codes

Code	Definition
@ERSUCC	Success
@ERIPVAL	Illegal parameter value

Example

```
 CHEKUM   TCOMPLET RETCODE=RCODE,TASKID=USRID
+CHEKUM   PUSHA 0,USRID             .SET task ID
+         OI    0(SP),X'80'
+         PUSHA 0,RCODE             .SET return code
+#TCOMPLT  STATIC
+         ORG   #TCOMPLT
+         DC    V(TCOMPLET)
+         CSECT
+         L     R1,=R(#TCOMPLT)
+         L     R1,0(R14,R1)
+         PUSH  0,R1
+         LA    R1,4(,SP)
+         JSI   0(,SP)
+         POPN  0,2*4+4

                 .
          (Static Section)
                 .
 RCODE    DS F
 USRID    DC F'0'
```

3.2.13 TINVOKE - Invoke Task

```
[label]  TINVOKE     RETCODE=retcode,
                     TIDLOC=tidloc,
                     EPLOC=eploc,
                     WS=ws
                    [,LIBRARY=library]
                    [,VOLUME=volume]
                    [,SYSTEM=system]
                    [,DATAREALTH=datarealth]
                    [,QUOTA=quota]
                    [,USER=user]
                    [,PASSWORD=password]
                    [,DISABHELP=disabhelp]
```

Function

Through the TINVOKE service, a running program can create another task. The new task is the child of the invoking program's task and can be the parent of other tasks through programs issuing TINVOKEs. The new task may either be an interactive task (i.e., foreground with an associated workstation) or a non interactive task that executes programs through procedures.

There is a limit to the number of subtasks that a task may create. The system maximum for a task is 255. For each subsequent TINVOKE within the parent-child chain, this quota may not exceed (QUOTA-1) of the parent task. In order for the parent task to regain its original quota, all subtasks must release all resources and be terminated.

If the newly created task is an interactive task and the EPLOC parameter is not specified, the task is created and control is passed to the command processor which displays the Command Processor menu. When a logoff command is received by the command processor either by pressing PF key 16 at the workstation or through a program issuing a call to the LOGOFF system service, the task will be removed from the system. The parent task must issue a TCOMPLET to insure that the subtask is finished. If the EPLOC parameter is specified on the interactive task invocation, the specified program will be initiated at the workstation.

If the newly created task is a background task, the procedure is executed and upon completion the task is removed from the system. The parent must check that the task has completed.

The parent task must insure that all its children are removed from the system before it can be terminated. See the TKILL and the TCOMPLET system services for how to remove tasks from the system.

Parameter Definitions

Parameter Definition   I/O   Data Type

datarealth   Input   fixed bin(31,0)

Specifies the data segment size for the task to be created. It must be a value between 64K and 8128K bytes. The default size is 256K bytes.

disabhelp   Input   fixed bin(15,0)

A value of 1 disables the HELP key. If the value is 0 and the USER parameter is not specified, TINVOKE uses the HELP setting of the calling program's task. If the value is 0 and the USER parameter is specified, TINVOKE uses the HELP setting of the specified user.

eploc   Input   char(8) var

Specifies the name of the program or procedure to run in the newly created subtask. If the WS parameter is not specified, the program runs in the background and the task is removed from the system when the program ends. If WS is specified, the program runs as an interactive task in the foreground and control passes to the command processor when the program is completed. This parameter is required if the WS parameter is not specified.

library   Input   char(8) var

Library to be searched for the program indicated by the EPLOC parameter. If EPLOC is specified, this parameter is required and the SYSTEM parameter may not be coded.

password   Input   char(8) var

Specifies the password for the USERID. This is a required parameter if the caller is not privileged and the USERID parameter is specified.

quota   Input   fixed bin(31,0)

Specifies the maximum number of subtasks which the new task can create. The default value is 0. The maximum value is 255. This quota may not exceed the parent task's (QUOTA-1).

retcode   Input   fixed bin(31,0)

Code that indicates the success or failure of the routine call.

system Input fixed bin(15,0)

A value of 1 indicates that the system defaults are to be used for the LIBRARY and VOLUME parameter values. A value of 0 indicates that the values specified with the LIBRARY and VOLUME parameters are to be used in the search for the program file. SYSTEM is a required parameter if EPLOC is specified.

tidloc Output fixed bin(31,0)

Specifies a storage location where the task number of the created task may be stored. This number is used as input to the CHECK, TCOMPLET, and KILL system services. A required parameter.

user Input char(8) var

Specifies the USERID under which the program is to be run. The subtask's base file access privileges are determined by this ID; the default is the same user ID as the task which is calling TINVOKE. If no program is specified via EPLOC, the LOGON procedure for the specified user ID is run. If supplied by a task which is not privileged, the PASSWORD parameter must also be supplied.

volume Input char(8) var

Specifies the volume name for the program to be run. This is a required parameter if EPLOC is specified. It may not be used with the SYSTEM parameter.

ws Input fixed bin(15,0)

Specifies the workstation number to associate with the task. Specifying the WS parameter indicates an interactive task and the program is to run in the foreground. The workstation must be reserved by the calling routine. On completion of the subtask the workstation is released. It can be retrieved by using CHECK or TCOMPLET. This option must be specified if EPLOC is not specified.

Return Codes

Code	Definition
@ERSUCC	Success.
@ERIPVAL	Illegal parameter value.
@ERNOTRES	Specified workstation is not reserved by caller.
@ERDATSEGSIZ	Invalid data segment size specified.
@ERUSRPW	Invalid user ID and/or password.
@ERUSRLST	Unable to read the userlist.
@ERTHEAP	GETMEM failure (including GETBLOK failure due to GETMEM).
@ERTASKCR	Unable to create task (GETBLOK failure other than GETMEM).
@ERURESWS	Specified user is restricted from this workstation.
@ERINSUFQ	Insufficient task quota to satisfy request.
@ERFDE	Program file specified does not exist.

Example

```
 MAKETSK  TINVOKE RETCODE=RCODE,TIDLOC=USRID,WS=WORKST,EPLOC=PROG,      -
                LIBRARY=MYLIB,VOLUME=MYVOL,DATAREALTH=DATASIZE,         -
                QUOTA=NUMTSK
+MAKETSK  PUSHA 0,NUMTSK           .Set Quota value
+         OI    0(SP),X'80'
+         PUSHA 0,DATASIZE         .SET SEG2LTH
+         PUSHA 0,0
+         PUSHA 0,MYVOL            .Set Volume
+         PUSHA 0,MYLIB            .Set Library
+         PUSHA 0,WORKST           .Set Workstation
+         PUSHA 0,PROG             .SET PROGRAM NAME
+         PUSHA 0,USRID            .SET TIDLOC
+         PUSHA 0,RCODE            .SET Return Code
```

```
+#TINVOKE STATIC
+         ORG   #TINVOKE
+         DC    V(TINVOKE)
+         CSECT
+         L     R1,=R(#TINVOKE)
+         L     R1,0(R14,R1)
+         PUSH  0,R1
+         LA    R1,4(,SP)
+         JSI   0(,SP)
+         POPN  0,9*4+4
                 .
          (Static Section)
                 .
 RCODE    DS F
 USRID    DC F'0'
 WORKST   DS H
 PROG     DC CL8'TAXPROG'
 MYLIB    DC CL8'PAYEES  '
 MYVOL    DC CL6'MONEY'
 DATASIZE DC F'512'
 NUMTSK   DC F'4'
```

3.2.14 TKILL - Task Termination

Syntax

```
[label] TKILL  RETCODE=retcode,
               TASKID=taskid
```

Function

This service allows a parent task to force a child task and all of the descendants into CANCEL and LOGOFF. All resources associated with the specified child and descendants are returned to the system. The issuing task must be the parent of the specified task.

Parameter Definitions

Parameter Definition	I/O	Data Type
retcode	Output	fixed bin(31,0)

Code that indicates the success or failure of the routine call.

Parameter Definition	I/O	Data Type
taskid	Input	fixed bin(31,0)

Specifies the task number of the subtask to be logged off and cancelled.

Return Codes

Code	Definition
@ERSUCC	Success.
@ERIPVAL	Illegal parameter value.
@ERNOTDESC	Specified subtask is not an immediate descendant of the caller.

Example

```
 KILLUM   TKILL RETCODE=RCODE,TASKID=USRID
+KILLUM   PUSHA 0,USRID             .SET task ID
+         OI    0(SP),X'80'
+         PUSHA 0,RCODE             .SET return code
+#TKILL   STATIC
+         ORG   #TKILL
+         DC    V(TKILL)
+         CSECT
+         L     R1,=R(#TKILL)
+         L     R1,0(R14,R1)
+         PUSH  0,R1
+         LA    R1,4(,SP)
+         JSI   0(,SP)
+         POPN  0,2*4+4
                .
          (Static Section)
                .
 RCODE    DS F
 USRID    DC F'0'
```

3.2.15 VOLINFO - Extract Volume Information

Syntax

```
[label]  VOLINFO  RETURNCODE=returncode,
                  VOLNAME=volname,
                  VSID=vsid
                 [,TYPE=type]
                 [,MOUNTER=mounter]
                 [,BC=bc]
                 [,MAXTFR=maxtfr]
                 [,CV=cv]
                 [,CVP=cvp]
                 [,CVD=cvd]
                 [,SECTYPE=sectype]
                 [,TOL=tol]
                 [,DEVNUM=devnum]
                 [,VCBADDR=vcbaddr]
```

Function

This service extracts system information on a specific disk.

Parameter Definitions

Parameter Definition	I/O	Data Type
bc	Output	fixed bin(15,0)

Returns the number of blocks per cylinder on this disk.

cv	Output	fixed bin(15,0)

Returns the number of cylinders per disk.

cvd	Output	fixed bin(15,0)

Returns the number of cylinders per diagnostic disk.

cvp	Output	fixed bin(15,0)

Returns the number of cylinders per physical disk.

devnum	Output	char(1)

Returns the device number on which the disk is mounted.

maxtfr	Output	fixed bin(15,0)

Returns the maximum number of bytes in a transfer.

mounter Output char(3)

Returns the user ID of the disk mounter.

returncode Output fixed bin (31,0)

Code that indicates the success or failure of the routine call.

sectype Output char(1)

Returns the sector type (diskette only). (S) indicates a soft sectored disk, (H) indicates a hard sectored disk.

tol Output char(2)

Returns the fault tolerance level. (CT) indicates crash tolerance, (MT) indicates media tolerance and ( ) indicates no tolerance.

type Output char(1)

Returns the disk type. F indicates fixed, R indicates removable disk, blank indicates disk not mounted.

vcbaddr Output fixed bin (31,0)

Returns the VCB address for this disk

volname Input char(8)

The name of the disk for which the information request applies.

vsid Input binary(8)

Volume set identification number of the disk for which the information request applies.

Return Codes

Code	Definition
@ERSUCC	Success
@ERVNM	Volume not mounted

Examples

```
 GETVOL   VOLINFO RETURNCODE=RCODE,VOLNAME=MYVOL,VSID=VOLNUM,           -
                MOUNTER=WHO,DEVNUM=DEVICE,VCBADDR=BLKNUM
+GETVOL   DS    0H
+         PUSHA 0,BLKNUM
+         MVI   0(SP),X'80'        flag last argument
+         PUSHA 0,DEVICE
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,WHO
+         PUSHA 0,0
+         PUSHA 0,VOLNUM
+         PUSHA 0,MYVOL
+         PUSHA 0,RCODE
+#VOLINF  STATIC
+         ORG   #VOLINF
+         DC    V(VOLINFO)
+         CSECT
+         L     R1,=R(#VOLINF)
+         L     R1,0(R14,R1)
+         PUSH  0,R1
+         LA    R1,4(,SP)
+         JSI   0(,SP)
+         POPN  0,14*4+4
                 .
          (Static Section)
                 .
 RCODE    DS F
 MYVOL    DC CL8'OFFICE'
 WHO      DS CL3
 VOLNUM   DS BL1
 DEVICE   DS CL1
 BLKNUM   DS F
```

3.2.16 VSETINFO - Extract Information about a Volume Set

```
[label]  VSETINFO  RETURNCODE=returncode,
                   VOLNAME=volname
                  [,SETTYPE=settype]
                  [,LABELTYP=labeltyp]
                  [,USAGE=usage]
                  [,USER=user]
                  [,OCNT=ocnt]
                  [,ADDREF=addref]
                  [,PAGE=page]
                  [,SPOOL=spool]
                  [,WORK=work]
                  [,SECURE=secure]
                  [,XLMTOPEN=xlmtopen]
                  [,XLMTTOTL=xlmttotl]
                  [,VSIDMAP=vsidmap]
                  [,ROOTMTD=rootmtd]
```

Function

This service extracts volume information on volume sets.

Parameter Definitions

Parameter Definition   I/O   Data Type

addref   Output   char(1)

Returns the addressing in effect. An S specifies standard and N specifies nonstandard.

labeltyp   Output   char(2)

Returns the volume label type. SL specifies standard label and NL specifies no label.

ocnt   Output   fixed bin(15,0)

Returns the number of open files on this volume.

page   Output   char(1)

Specifies whether or not paging files are allowed on the volume. Y specifies yes, N specifies no.

returncode   Output   fixed bin(31,0)

Code that indicates the success or failure of the routine call.

rootmtd Output char(1)

Returns whether the root volume of the volume set is mounted or not. Y indicates the root volume is mounted. N indicates that it is not. For a single volume, this ROOTMTD is N.

secure Output char(1)

Returns whether this is a secure volume set or not. Y indicates volume set is secure. N indicates that it is not.

settype Output char(1)

Returns the volume set type; S indicates a single volume, M indicates a volume set.

spool Output char(1)

Returns whether the volume is eligible for spool files. Y indicates that spool files are allowed, N indicates that they are not allowed on the volume.

usage Output char(2)

Returns the volume usage. SH indicates the volume is opened in shared mode, RR indicates restricted removal, PR indicates protected, EX indicates exclusive, or the field may be blank.

user Output char(3)

Returns the user ID of the volume user.

volname Input char(8)

Specifies the name of the volume for which the information request applies. Required parameter.

vsidmap Output char(32)

Returns a 32-byte bitmap showing the VSIDs of the mounted volumes of a volume set. Valid only for volume sets.

work Output char(1)

Returns whether the volume is eligible for work files. Y indicates that work files can be stored on the volume, N indicates that they can not.

xlmtopen   Output   fixed bin(15,0)

Returns the maximum number of extents allowed on opening a file on this volume.

xlmttotl   Output   fixed bin(31,0)

Returns the total extent limit for the volume set.

Return Codes

Code	Definition
@ERSUCC	Success
@ERVNM	Volume not mounted

Examples

```
 GETINFO  VSETINFO RETURNCODE=RCODE,VOLNAME=MYVOL,USER=WHO,PAGE=PG,      -
                ROOTMTD=VOLROOT,LABELTYP=LABEL
+GETINFO  DS    0H
+         PUSHA 0,VOLROOT
+         MVI   0(SP),X'80'        flag last argument
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,PG
+         PUSHA 0,0
+         PUSHA 0,0
+         PUSHA 0,WHO
+         PUSHA 0,0
+         PUSHA 0,LABEL
+         PUSHA 0,0
+         PUSHA 0,MYVOL
+         PUSHA 0,RCODE
+#VSETINF STATIC
+         ORG   #VSETINF
+         DC    V(VSETINFO)
+         CSECT
+         L     R1,=R(#VSETINF)
+         L     R1,0(R14,R1)
+         PUSH  0,R1
+         LA    R1,4(,SP)
+         JSI   0(,SP)
+         POPN  0,16*4+4
                .
```

```
          (Static Section)
                 .
RCODE     DS F
MYVOL     DC CL8'OFFICE'
WHO       DS CL3
PG        DS C
VOLROOT   DS C
LABEL     DS CL2
```

3.3 PROGRAMMING EXAMPLES

This section contains three programming examples using the memory management, security, and user synchronization system services. These programs also contain examples of using system services described in Chapter 4.

NOTE

The example programs in this section are provided to assist users in preparation of their own programs. They are not supported Wang products.

3.3.1 Memory Management Example

```
*THIS PROGRAM IS INTENDED AS A DEMONSTRATION OF MSMAP.
*THE PROGRAM MAPS A SINGLE DATA FILE INTO IT'S ADDRESS SPACE.
*THE FILE CONTAINS AN ARRAY OF 100 INTEGERS.
*THE PROGRAM MERELY COMPUTES THE SUM OF THESE INTEGERS.
R0       EQU    0
R1       EQU    1
R2       EQU    2
R3       EQU    3
DB       EQU    12
CB       EQU    13
R14      EQU    14
SP       EQU    15
CMP      CODE
         PRINT  NOGEN
BEGIN    BALR   CB,0                  BASE
         USING  *,CB
         L      DB,=R(DMP)
         AR     DB,R14
         USING  DMP,DB
         EXTRACT INVOL=PVOL,INLIB=PLIB
*THE FOLLOWING CALL MAPS AN EXISTING DATA FILE AT
*ANY AVAILABLE ADDRESS
         MSMAP  RETURNCODE=RC,
                PATHNAME=PTH,
                TYPE==Y(2),
                OPTION==Y(0),
                COMMAND==Y(0),
                STRTADDR=SA
         LT     R0,RC                 ANY ERRORS?
         BNZ    DIE                   YES
         L      R1,SA                 ADDRESS OF DATA
         LA     R2,100                NUMBER OF WORDS OF DATA
         XR     R3,R3                 ZERO
```

```
LOOP     A     R3,0(R1)            ADD NUMBER TO SUM
         LA    R1,4(R1)            ADDR OF NEXT WORD
         BCT   R2,LOOP             AGAIN
         ST    R3,SUM              STORE SUM
*THE FOLLOWING CALL UNMAPS THE FILE.
*THIS ISN'T REALLY NECESSARY AS THE UNLINK WOULD DO IT ANYWAY
DONE     MSUNMAP RETURNCODE=RC,
                 PATHNAME=PTH
         LT    R0,RC               ANY ERRORS?
         BNZ   DIE                 YES
         L     R0,SUM              RETURN THE SUM
         RT
DIE      DC    Y(0)                ENTER DEBUGGER
         LTORG
               =R(DMP)
               =R(#MSMAP)
               =R(#MSUNMAP)
               =Y(0)
               =Y(2)
DMP      STATIC
RC       DS    A                   RETURN CODE
SA       DS    A                   ADDRESS WHERE FILE WAS MAPPED
SUM      DS    A                   THE SUM
PTH      DC    Y(6+8+8)            FILESPEC:
PVOL     DS    CL6                 VOLUME
PLIB     DS    CL8                 LIBRARY
PFIL     DC    CL8'MPDAT'          FILENAME
         END   BEGIN
```

3.3.2 Security Logging Example

```
**********************************************************************
*                                                                    *
*  THIS PROGRAM READS LOG SETTING PARAMETERS FROM THE WORKSTATION    *
* AND TRANSLATES THE HEX CHARACTERS INTO THEIR BINARY EQUIVALENT. IT *
* THEN USES THE BINARY REPRESENTATION OF THE INPUT AS ARGUMENTS TO   *
* THE "CNTRLOG" MACRO WHICH UPDATES THE SECURITY LOGGING OF THE      *
* SYSTEM.                                                            *
*                                                                    *
*  THE OPTIONS OF THIS PROGRAM ALSO INCLUDE THE ABILITY TO STOP      *
* LOGGING, RESUME LOGGING, CHANGE THE LOG PARAMETERS AND CONTINUE    *
* LOGGING, AS WELL AS CREATE A NEW LOG.                              *
*                                                                    *
* "HEXCHAR" IS THE NUMBER OF HEX CHARACTERS THAT REPRESENT THE BIT   *
*  STRING USED BY "CNTRLOG" TO SET THE LOGGING PARAMETERS.           *
*  WHEN MORE EVENTS ARE ADDED TO THE LOGGING                         *
*  CAPABILITY, "HEXCHAR" MUST BE CHANGED FOR THIS PROGRAM TO BE RUN  *
*  PROPERLY. CURRENTLY, THIS PROGRAM CAN BE USED TO SET 32 EVENTS    *
*  AND VIOLATIONS ("HEXCHAR"*4 BITS). "HEXCHAR" MUST BE A MULTIPLE OF*
*  FOUR.                                                             *
**********************************************************************
```

```
         REGS
HEXCHAR  EQU   8                         NO. HEX CHARACTERS
STOPLOG  EQU   65                        STOP LOGGING OPTION
STRTNEW  EQU   66                        START LOG , NEW PARAM
STRTOLD  EQU   67                        START  LOG ,OLD PARAM
RSMNEW   EQU   68                        RESUME LOG NEW PARAMETERS
RSMOLD   EQU   69                        RESUME LOG , OLD PARAMETERS
CHGEVTS  EQU   70                        CHANGE LOG EVENTS
STPCNTL  EQU   1                         STOP LOGGING CONTROL PARAM
NEWCNTL  EQU   2                         START NEW LOG CONTROL PARAM
RSMCNTL  EQU   3                         RESUME LOG CONTROL PARAM
         BALR  EP,0
         USING *,EP
         LR    R12,R14
         AL    R12,=R(TESTSTAT)
         USING TESTSTAT,R12
*
*
         L     R10,KEYS                  LOAD THE PFKEY MASK
         GETPARM FORM=SELECT,KEYLIST=CNTRL,MSG=MSG1,PFKEYS=(R10)
*
*
* GET THE PFKEY NUMBER IN HEX AND STORE
         LC    R4,CNTRL+8
         ST    R4,PFKEY
*
*
* CHECK FOR STOP,RESUME WITH OLD PARAM., START WITH NO PAR. CHANGE
         L     R4,PFKEY
         LA    R5,STOPLOG
         CR    R4,R5
         BE    STOPRES
         LA    R5,RSMOLD
         CR    R4,R5
         BE    STOPRES
         LA    R5,STRTOLD
         CR    R4,R5
         BE    NEWFIL
*
*
* NEED BIT SETTINGS FOR NEWLOG, CHANGE OPTIONS, AND RESUME NEW SETTING
         GETPARM KEYLIST=SETLOG,MSG=MSG2  GET EVENT+VIO. SETTINGS
************************************************************************
*                                                                      *
* CALCULATE THE NUMBER OF TIMES TO EXECUTE  LOOP1.                     *
* THE NUMBER OF TIMES = HEXCHAR/4. THIS IS BECAUSE THE REGISTER        *
* CAN HOLD ONLY FOUR CHARACTERS AT A TIME.                             *
************************************************************************
*
```

```
         LA    R10,0                      PREPARE FOR DIVIDE
         LA    R11,HEXCHAR
         D     R10,=F'4'                  DIVIDE BY 4
         ST    R11,LPCOUNT                STORE THE LPCOUNT FOR THE
*                                            2ND PASS FOR VIOLATIONS
*
*
*
         LA    R1,SETBITS                 GET ADDRESS OF PROPER BIT
*                                           STRG 1ST PASS = EVENTS
*                                           2ND PASS = VIOLATIONS
         LA    R2,INPARM                    GET ADDRESS OF PROPER INPUT
*                                           1ST PASS = EVENT SET INPUT
*                                           2ND PASS = VIOL. SET INPUT
*
*
         LA    R3,2                       LOAD NUMBER OF PASSES TO LOOP
*                                           CONTROL REGISTER
*
LOOP     L     R10,LPCOUNT               GET THE LOOP CONTROL FOR LOOP1
         L     R7,0(0,R1)                R7 =PARAMETER BIT STRG ADDRESS
         L     R11,0(0,R2)               R11 = INPUT STRG ADDRESS
         LA    R11,12(0,R11)             ADD 12 BECAUSE OF GETPARM
         TR    0(HEXCHAR,R11),TRTAB      TRANSLATE HEX INPUT TO BINARY
*
***********************************************************************
*                                                                     *
* AFTER THE NEXT INSTRUCTION, R5 WILL CONTAIN VVALID BITS EVERY FOUR  *
* PLACES STARTING WITH THE FOURTH BIT.                                *
*                                                                     *
*  E.G.                                                               *
*  SCREEN INPUT = AAAAAAAA                                            *
*  AFTER EXECUTION R5=0A0A0A0A                                        *
*  THE ZEROES MUST BE STRIPPED OFF AND THE REMAINING 16 BITS MUST     *
*  BE STORED IN THE APPROPRIATE PLACE(R7 POINTS) IN THE BIT STRG      *
*            ARGUMENT
************************************************************************
*
LOOP1    L     R5,0(0,R11)                LOAD 4 CHARACTERS (32 BITS)
         LA    R0,4                       R0= LOOP2 CONTROL
LOOP2    SLL   R5,4                       STRIP BITS OFF
         SLDL  R4,4                       MOVE GOOD BITS TO R4
         BCT   R0,LOOP2                   DO FOUR TIMES
*
         ST    R4,TEMP                    STORE THE DATA
         MVC   0(2,R7),TEMP+2             MOVE THE LAST TWO BYTES TO
*                                           APPROPRIATE BIT STRG
         LA    R7,2(0,R7)                 INCREMENT BIT STRG PTR
         LA    R11,4(0,R11)               INCREMENT INPUT STRG PTR
         BCT   R10,LOOP1
         LA    R1,4(0,R1)                 R1 NOW PTS TO VIOL. BIT STRG
         LA    R2,4(0,R2)                 R2 NOW PTS TO VIOL. INPUT STR
         BCT   R3,LOOP
*
```

```
* THIS ENSURES THAT EVERYTHING IS INITIALLY TURNED OFF
         LA    R4,NOEVENT
         MVI   0(R4),X'FF'
         MVC   1(31,R4),0(R4)
         LA    R4,NOVIOL
         MVI   0(R4),X'FF'
         MVC   1(31,R4),0(R4)
*
*
* IF THE OPTION  IS TO JUST CHANGE BIT SETTINGS OR RESUME WITH NEW BIT
*    SETTINGS THEN BRANCH AROUND LIBRARY AND VOLUME
*
         L     R4,PFKEY
         LA    R5,CHGEVTS
         CR    R4,R5
         BE    CHGSET
         LA    R5,RSMNEW
         CR    R4,R5
         BE    NEWRES
*
* GET THE LIBRARY AND VOLUME
NEWFIL   GETPARM KEYLIST=INPUT,MSG=MSG3   GET LIBRARY AND VOLUME
*
*
* DETERMINE THE LENGTH OF THE VOLUME NAME AND STORE IT
         LA    R1,6                       SIX POSSIBLE CHARACTERS
         LA    R5,0                       R5=NO. CHAR IN VOL. NAME
         LA    R4,VOLUME+12               GET THE START POSITION
         LA    R9,VOLLEN+2                GET THE START OF VOLUME
*                                          PARAMETER TO BE PASSED
LOOP3    CLI   0(R4),X'20'                LOOK FOR BLANK
         BE    ENDOFVL                    FOUND END OF STRG
         MVC   0(1,R9),0(R4)              MOVE NON BLANK CHAR.
         LA    R9,1(0,R9)                 INC. PARAMATER POSITION
         LA    R4,1(0,R4)                 GET THE NEXT BYTE
         LA    R5,1(0,R5)                 INCREMENT STRING COUNT
         BCT   R1,LOOP3
ENDOFVL  STH   R5,VOLLEN                  STORE THE NO. CHAR
*
* GET THE NUMBER OF CHAR. IN LIBRARY NAME AND STORE
         LA    R1,8                       EIGHT POSSIBLE CHARACTERS
         LA    R5,0                       R5=NO. CHAR IN LIB. NAME
         LA    R4,LIBRARY+12              GET THE START POSITION
         LA    R9,LIBLEN+2                GET THE START OF LIBRARY
*                                          PARAMETER TO BE PASSED
LOOP4    CLI   0(R4),X'20'                LOOK FOR BLANK
         BE    ENDOFLB                    FOUND END OF STRG
         MVC   0(1,R9),0(R4)              MOVE NONBLANK CHAR.
         LA    R9,1(0,R9)                 INC. PARAMETER POSITION
         LA    R4,1(0,R4)                 GET THE NEXT BYTE
         LA    R5,1(0,R5)                 INCREMENT STRING COUNT
         BCT   R1,LOOP4
ENDOFLB  STH   R5,LIBLEN                  STORE THE NO. CHAR
*
```

```
* CHECK FOR NEW LOG FILE , OLD PARAMETERS
         L     R4,PFKEY
         LA    R5,STRTOLD
         CR    R4,R5
         BE    NEWOLD
*
*
*  NEWLOG, NEW PARAMETERS
         LA    R5,NEWCNTL                 GET THE NEW CONTROL PAR.
         ST    R5,LOGCNTL                 STORE IT IN THE CONTROL
CREATE   CNTROLOG RC=RC,SETEVENTS=SETEVTS,
               SETVIOLATION=SETVIOS,CONTROL=LOGCNTL,
               RESETEVENTS=NOEVENT,RESETVIOLATION=NOVIOL,
               NEWLIB=LIBLEN,NEWVOL=VOLLEN
         B     DONE
*
* THIS HANDLES THE NEWLOG WITH OLD PARAMETERS
NEWOLD   LA    R5,NEWCNTL
         ST    R5,LOGCNTL
         CNTROLOG RC=RC,CONTROL=LOGCNTL,NEWLIB=LIBLEN,NEWVOL=VOLLEN
         B     DONE
*
* THIS WORKS FOR THE STOP AND RESUME WITH SAME PARAMETERS
STOPRES  L     R5,PFKEY                   GET  PFKEY
         LA    R4,STOPLOG                 GET STOPLOG PFKEY
         CR    R5,R4                      CHECK FOR STOP LOG
         BE    STOPKEY                    PROPER CONTROL IN PRESENT
         LA    R5,RSMCNTL                 GET THE RESUME CONTROL
         ST    R5,LOGCNTL                 STORE IN LOG CONTROL
         B     RESTOP
STOPKEY  LA    R5,STPCNTL
         ST    R5,LOGCNTL
*
* MAKE CALL FOR RESUME WITH SAME PARAMS. AND STOP
RESTOP    CNTROLOG RC=RC,CONTROL=LOGCNTL
         B     DONE
*
*
* MAKE CALL FOR RESUME WITH NEW PARAMETERS
NEWRES   LA    R5,RSMCNTL
         ST    R5,LOGCNTL
         CNTROLOG RC=RC,SETEVENTS=SETEVTS,RESETVIOLATION=NOVIOL,       +
               SETVIOLATION=SETVIOS,RESETEVENTS=NOEVENT,CONTROL=LOGCNTL
         B     DONE
*
```

```
* THIS CHANGES BIT SETTINGS
CHGSET   CNTROLOG RC=RC,SETEVENTS=SETEVTS,RESETVIOLATION=NOVIOL,       +
               SETVIOLATION=SETVIOS,RESETEVENTS=NOEVENT
DONE     RT
TESTSTAT STATIC
RC       DC F'0'
LOGCNTL  DC F'0'                         SET NEWLOG PARAMETER
PFKEY    DS F
LIBLEN   DS H                            NEW LOG LIBRARY
         DS CL8
VOLLEN   DS H                            NEW VOLUME LIBRARY
         DS CL6
SETEVTS  DC BL.256'0'                    LOG EVENTS BIT STRG
SETVIOS  DC BL.256'0'                    LOG VIOLATIONS BIT STRG
NOEVENT  DC BL.256'0'                    TURN OFF EVENTS BIT STRG
NOVIOL   DC BL.256'0'                    TURN OFF VIOLATIONS BIT STRG
INPUT    KEYLIST PRNAME='INPUT',LABELPFX='',PREVIEW=YES,               +
               'VOLUME',('WORK  ',ANL),'LIBRARY',('@SYSLOG@',ANL)
SETLOG   KEYLIST PRNAME='LOGHEX',LABELPFX='',PREVIEW=YES,              +
               'LOGSET',('00000000',HEX),'VIOSET',('00000000',HEX)
CNTRL    KEYLIST PRNAME='LOGCNTL',LABELPFX='',PREVIEW=YES,TEXT,('PF1   +
                STOP LOGGING',1,'A10'),TEXT,('PF2    START LOGGING IN A+
                NEW FILE WITH NEW PARAMETERS',1,'A10'),TEXT,('PF3    ST+
               ART LOGGING IN A NEW FILE WITH THE OLD PARAMETERS',1,'A1+
               0'),TEXT,('PF4    RESUME LOGGING IN THE LAST FILE WITH N+
               EW PARAMETERS',1,'A10'),TEXT,('PF5    RESUME LOGGING IN +
               THE LAST FILE WITH THE OLD PARAMETERS',1,'A10'),TEXT,('P+
               F6    CHANGE THE LOGGING PARAMETERS ONLY',1,'A10')
         DS 0F                             AALLIGNMENT
KEYS     DC XL4'FC000000'
MSG3     MSGLIST '03','SCLOGT','ENTER THE LOGGING VOLUME AND LIBRARY'
MSG2     MSGLIST '02','SCLOGT','ENTER LOGGING AND VIOLATION BIT SETTING+
               S IN HEX'
MSG1     MSGLIST '01','SCLOGT','SELECT THE PF-KEY DESIRED'
*
TRTAB    DC CL256'0'                     TRANSLATE TABLE USED FOR
         ORG TRTAB+X'30'                 CONVERTING HEX TO BINARY
         DC X'00010203040506070809'
         ORG TRTAB+X'41'
         DC X'0A0B0C0D0E0F'
         ORG
TEMP     DC F'0'
SETBITS  DC A(SETEVTS)                   PTR TO LOG EVENTS BIT STRG
         DC A(SETVIOS)                   PTR TO LOG VIOLATIONS BIT STRG
INPARM   DC A(LOGSET)                    PTR TO LOG EVENTS INPUT
         DC A(VIOSET)                    PTR TO LOG VIOL. INPUT
LPCOUNT  DS F
         END
```

3.3.3 User Synchronization Example

This program is intended to show how to use the user synchronization services.

- First, use SCREATE to create the object for synchonization.

- To access the resource that is being managed, use SENTER. SENTER releases the resource to the program once its free. At this point, the application program would be able to process the resource as needed.

- SEXIT removes the synchonization object from the program and activates the next request for the object.

- To delete the synchronization object, you have to enter it (use SENTER). Then, use SDELETE to remove it and free the associated memory space. If there are other requests to use the resource when the program issues SDELETE, all requests are processed before SDELETE is run.

```
START    CODE
         REGS
         BALR  EP,0
         USING *,EP
         USING DATA,R14
         SCREATE RETURNCODE=RC,HANDLES=PTR
         LT    R1,RC           CHECK RETURN CODE
         BNZ   CRTERROR        BRANCH IF ERROR
         SENTER  RETURNCODE=RC,HANDLES=PTR
         LT    R1,RC           CHECK RETURN CODE
         BNZ   ENTERROR        BRANCH IF ERROR
         SEXIT    RETURNCODE=RC,HANDLES=PTR
         LT    R1,RC           CHECK RETURN CODE
         BNZ   EXTERROR        BRANCH IF ERROR
         SENTER  RETURNCODE=RC,HANDLES=PTR
         LT    R1,RC           CHECK RETURN CODE
         BNZ   ENTERROR        BRANCH IF ERROR
         SDELETE RETURNCODE=RC,HANDLES=PTR
         LT    R1,RC           CHECK RETURN CODE
         BNZ   DELERROR        BRANCH IF ERROR
         RT
CRTERROR RT
ENTERROR RT
EXTERROR RT
DELERROR RT
DATA     STATIC
PTR      DC    A(HANDLE)
HANDLE   DS    CL8
RC       DS    F               RETURN CODE
         END
```

CHAPTER 4
SVC-TYPE SERVICES AND ASSOCIATED MACROINSTRUCTIONS

4.1 OVERVIEW

This chapter describes the system services available for general use that are invoked by issuing an SVC instruction. The assembler interface to system services are macros located in the system library @MACLIB@ on the system volume, which the assembler accesses when assembling a source program.

In the following sections, each system service description contains the following information:

- Syntax -- This section describes the format in which to code a macroinstruction. There may be more than one possible format. The programmer must adhere to assembly language syntax rules as described in the VS Assembly Language Reference when coding the macroinstructions.

- Function -- This section describes the functions of each macroinstruction.

- Parameter Definitions -- This section describes in detail the parameters that may be used with the macro call, and the valid values for each parameter.

- Structure -- When present, this section describes system control blocks in graphic form showing the offsets (in hexadecimal) for each symbol in the control block.

- Output -- This section describes the output of the SVC, including the information placed on or removed from the program stack and the valid return codes for the SVC. This section is omitted for those macroinstructions that generate or describe system data structures.

In cases where there are restrictions on the use of the macroinstruction, a separate section is included that describes these restrictions.

4.2 SERVICE-BY-SERVICE DESCRIPTIONS

Macroinstructions described here are for two commonly performed operations: the description of a system control block and the generation of an orderly call to the supervisor to perform a service.

Corresponding to each system control block is a macroinstruction which system and user programs freely use to define standard labels for fields within the control block. If only the macroinstruction name is coded, the system generates a dummy section (DSECT) of that name. If a register specification is included, a USING instruction is also generated. If the user provides a SUFFIX parameter, each label generated contains the suffix character immediately following the block name. (The suffix must be one character only.) If the user specifies the NODSECT parameter, the DSECT statement is not generated.

4.2.1 AXD1 - Describe AXD1 Structure

Syntax

```
    AXD1    [NODSECT][,REG=expression][,SUFFIX=character]
```

Function

Allows the user to symbolically reference the Alternate Descriptor Block (AXD1) which describes the alternate index structure of an indexed file. An indexed file has an AXD1 block if, and only if, a flag (FDR1FLAGSALTX) is set in its label (FDR1).

Parameter Definitions

NODSECT Specification of NODSECT results in the AXD1 fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, a DSECT with the name AXD1 (plus the optional suffix) is generated.

REG Provides for the optional specification of a register for which a USING statement for the AXD1 fields is generated.

SUFFIX If provided, all labels are generated by the concatenation of the letters AXD1, the user-provided SUFFIX (one ASCII character in length), and the field name.

Structure

		BYTE 0	BYTE 1	BYTE 2	BYTE 3	
AXD1 BEGIN	+0	BL				
	+4	MASK				
	+8					
	+C	UFB				
	+10	ALTINX	FLAGS	MSIZE	DUPINX	
	+14	BCB				
	+18					
	+1C					
	+20					
	+24	PMASK				
	+28					
	+2C	ORECSIZE		OFLAGS	OSTART	
	+30			ONRECS		
	+34		OEBLK			
	+38	OSPAREX		OSPARE		
ENTRY	+3C	XORD	EFLAGS	XLEVELS		
	+40	KEYPOS		KEYSIZE	HXBLK	
	+44			NRECS		
	+48		PTRD			
	+4C	PRLEN	PRAKPOS	PRPKPOS	ESPARE	
	+50					
	+54					
	+58					
				. . .		
	+73C	SPARE3				LENGTH 800

For DMS Processing

+2C	SAVEADR			
+30	SAVELTH			

For Save Area Type S

+2C	KEYSIZE	HXBLK		
+30	SEREC		ENTOFF	
+34	PTRN			CURINX
+38	SPAREX		EXSPARE	

Example

```
         AXD1  NODSECT
*,* AXD1 DEFINITION
***********************************************************************
*
*        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    07/16/82
*        VERSION 5.04.02
*
**********************************************************************
* BLOCK DESIGNATOR AREA:
AXD1BEGIN                  DS  0F
AXD1BL                     DS  BL4  BLOCK TYPE DESIGNATION
*                                   AXD1BL MUST EQUAL XL4'2'
*                                   OR XL4'4'

* DMS PROCESSING AREA:
AXD1MASK                   DS  BL8  BITS ON INDICATE ALTERNATE
*                                   INDEX STRUCTURES (NUMBERED
*                                   1 TO 16) PRESENT
*                                   (INITIAL IMPLEMENTATION OF
*                                   2-BYTE MASK ONLY)
AXD1UFB                    DS  A    POINTER TO UFB FOR THIS FILE
*                                   AFTER THE FILE HAS BEEN OPENED
AXD1ALTINX                 DS  BL1  ORDINAL INDEX NUMBER FOR READ
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)
AXD1FLAGSOPENA             EQU X'08' OPEN ALLOCATED THIS AXD1 BLOCK
*                                    (ONLY IF NOT OUTPUT MODE)
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
**
```

```
AXD1MSIZE                DS  BL1  SIZE OF MASK PER FILE
*                                 VALUE FROM 2-8 BYTES (MUST BE 2
*                                 FOR FIRST IMPLEMENTATION)
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.
*
AXD1BCB                  DS  BL16 BCB FOR DMS PROCESSING (SEE UFB)
AXD1PMASK                DS  BL8  MASK OF VALID ALTERNATE ACCESS
*                                 PATHS (SET AT FILE CREATION ONLY)
*
* THE FOLLOWING FIELDS ARE INTERMEDIATE OUTPUT MODE FIELDS
*
AXD1ORECSIZE             DS  H    WORK RECORD - MAX LENGTH
AXD1OFLAGS               DS  BL1  OUTPUT FLAGS (RESERVED)
AXD1OSTART               DS  BL3  FIRST BLOCK CONTAINING WORK RECORDS
AXD1ONRECS               DS  BL3  TOTAL COUNT OF WORK RECORDS
AXD1OEBLK                DS  BL3  LAST USED BLOCK NUMBER IN PRIMARY
*                                   TREE (ALT-TREE TO AXD1EBLK+1)
AXD1OSPAREX              DS  H    **** (unused) ****
AXD1OSPARE               DS  BL2  RESERVED IN OUTPUT MODE
**
              ORG AXD1ORECSIZE
* THE FOLLOWING FIELDS ARE USED FOR DMS PROCESSING (EXISTING FILES)
**
AXD1SAVEADR             DS   A  SAVE AREA ADDRESS (TYPE V)
AXD1SAVELTH             DS   H  SAVE AREA LENGTH (TYPE V)
              ORG AXD1ORECSIZE
* THE FOLLOWING 3 FIELDS ARE USED FOR SAVE AREA TYPE S
AXD1SKEYSIZE            DS   BL1  SAVED PRIMARY KEYSIZE
AXD1SHXBLK              DS   BL3  SAVED PRIMARY ROOT BLOCK NUMBER
AXD1SEREC               DS   H    SAVED PRIMARY LEVEL COUNT

*
AXD1ENTOFF              DS   H    OFFSET OF ACTIVE AXD1ENTRY(IN AXD1)
AXD1PTRN                DS   BL3  NEXT SEQUENTIAL BLOCK (ALT-TREE)
AXD1CURINX              DS   BL1  ORDINAL NUMBER ASSOCIATED WITH
*                                 BLOCK IN AXD1BCB
AXD1SPAREX              DS   H    **** (unused) ****
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:
AXD1ENTRY                  DS   0XL28  UP TO 64 ENTRIES
*                                      (EACH A DESCRIPTION OF ONE
*                                      ALTERNATE INDEX STRUCTURE;
*                                      UNUSED ENTRIES ZERO-FILLED)
AXD1XORD                   DS   HL1    ORDINAL NUMBER (STARTING FROM 1)
*                                      IDENTIFYING THIS INDEX STRUCTURE
*                                      (CORRESPONDS TO BIT IN
*                                      (AXDIMASK)
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
AXD1XLEVELS                DS   H      NUMBER OF LEVELS OF THIS
*                                      ALTERNATE INDEX STRUCTURE
*                                      EXCLUDING LOWEST LEVEL
AXD1KEYPOS                 DS   H      KEY POSITION IN RECORD
AXD1KEYSIZE                DS   HL1    KEY LENGTH
AXD1HXBLK                  DS   FL3    BLOCK-IN-FILE OF ROOT BLOCK
*                                      OF THIS ALTERNATE INDEX
AXD1NRECS                  DS   BL3    ITEM COUNT - LOW LEVEL OF TREE
AXD1PTRD                   DS   FL3    FIRST BLOCK OF LOW LEVEL
*                                      OF THIS ALTERNATE INDEX
*                                      (ALTERNATE KEY SEQUENCE)
AXD1PRLEN                  DS   BL1    LENGTH OF ALT TREE PSEUDO-REC
AXD1PRAKPOS                DS   BL1    POS OF ALT KEY IN PSEUDO-REC
```

```
AXD1PRPKPOS                 DS   BL1    POS OF PRI KEY IN PSEUOD-REC
AXD1ESPARE                  DS   BL9    (RESERVED IN EACH ENTRY)
AXD1ENTRYEND                EQU *
AXD1ENTRYLENGTH             EQU AXD1ENTRYEND-AXD1ENTRY
*
                   ORG AXD1ENTRY+64*L'AXD1ENTRY
AXD1SPARE3                  DS   XL196  (RESERVED)
*
AXD1END                     EQU *
AXD1LENGTH                  EQU AXD1END-AXD1BEGIN
```

4.2.2 AXDGEN - Generate Alternate Index Descriptor Block, (AXD1)

Syntax

```
[label]  AXDGEN [MASKSIZE={integer}][,ENTRIES={integer}]
                          {   2   }           {   0   }

                [,(ORD=integer,KEYPOS=integer,KEYSIZE=integer

                [,NODUPS][,COMPRESS])] ...
```

Function

Generates an alternate index descriptor block (AXD1) to be addressed by UFB field UFBALTPTR (ALTAREA parameter of UFBGEN macroinstruction). The AXD1 describes the alternate index structures of an indexed file. The AXD1 block contains up to 16 alternate index descriptions (AXD1ENTRY). Unused entries are filled with zeroes.

For existing files, no fields in the AXD1 are user-supplied prior to issuing the OPEN SVC. For files in Output mode, the user program may define each alternate index structure by supplying the access mask size, the key position, the key size, the flags, and the ordinal number of the index structure.

Parameter Definitions

MASKSIZE — The size, in bytes, of the alternate mask field. A bit is set in the mask that corresponds to the index of the alternate index structure. Must be specified as an integer. Must be equal to 2 for the current versions of the system. The parameter defaults to a value of 2.

ENTRIES — To use the AXD1 for OUTPUT mode processing, this parameter must equal the number of alternate index structures which are described in the following positional parameters. The value must be an integer from 0 to 16, and if not supplied, defaults to 0.

ORD — Ordinal number defining this alternate index structure (access path). This number corresponds to the On bit in the access mask. Specified as an integer between 1 and 16. Required in all supplied positional parameters.

KEYPOS — Key position in the record (i.e., offset in bytes into the record, counting from 0 for the first byte). Specified as an integer. Required in all supplied positional parameters.

KEYSIZE — Key length. Specified as an integer. Required in all supplied positional parameters.

NODUPS If specified for Output mode, duplicate keys are not allowed. The default is to allow duplicate keys. Ignored in other modes.

COMPRESS Ignored in the current system releases.

Example

```
          AXDGEN ENTRIES=1
+         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
```

4.2.3 BCE - Describe Buffer Control Entries

Syntax

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

Function

Describes the buffer control entries (BCE) which are contained in the buffer control table (BCTBL). There is one BCE per 2K buffer in a data management buffer pool.

Parameter Definitions

NODSECT — Specification of NODSECT results in the BCE fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, the system generates a DSECT with the name BCE (plus the optional SUFFIX).

REG — If specified, a USING statement is generated with the given register number.

SUFFIX — One ASCII character in length. If provided, all labels are generated by the concatenation of the letters BCE, the user-provided SUFFIX, and the field name.

Structure

BCE
BEGIN

		BYTE 0	BYTE 1	BYTE 2	BYTE 3	
	+0	OFB				
BUFCMD	+4	BUFADR				
	+8	BUFDATAL		SPARE		
	+C	BUFBLOCK			BCBFLAGS	
	+10	KEYHI				
	+14					
	+18					
	+1C	TYPE	WT	AGEWT	SPARE1	
	+20	IOCHN				
	+24	KEYLOW				
	+28					
	+2C					
	+30	EXPAND				
	+34					LENGTH = 38

Example

```
         BCE  REG=4
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
*
BCEBEGIN                   DS   0F           (FULLWORD ALIGNMENT)
BCEOFB                     DS   A            OFB ADDRESS
BCEBUFCMD                  DS   0BL1         COMMAND BYTE
BCEBUFADR                  DS   A            BUFFER MEMORY ADDRESS
BCEBUFDATAL                DS   H            IO-LENGTH (2K)
BCESPARE                   DS   H            OFFSET (UNUSED IN BCE)
BCEBUFBLOCK                DS   FL3          BLOCK WITHIN
*                                            FILE OF BUFFERED DATA
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
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.)
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
```

```
BCEWT                      DS   BL1         STARTING WEIGHT VALUE
BCEAGEWT                   DS   BL1         AGED WEIGHT VALUE
BCEFLAGS1                  DS   BL1         EXTRA FLAGS
BCEFLAGS1CLAIMED           EQU X'01'        REPL BCE HAS BEEN CLAIMD
BCEIOCHN                   DS   A           CHAIN FOR BCE'S WITH I/O
*                                           IN PROGRESS
BCEKEYLOW                  DS   CL12        TRUNCATED LOW KEY VALUE
*                                           (TYPE D)
BCEEXPAND                  DS   BL8         BCE EXPANSION
* EXPANSION = 12 (TRUNC KEYS =12), PLUS 4 (CHN BCE PER UFB)+4 EXTRA
BCELENGTH          EQU *-BCEBEGIN           BCE LENGTH (=56)
         CSECT
         USING BCE,4
```

4.2.4 BCTBL - Describe Buffer Control Table

Syntax

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

Function

Describes the buffer control table (BCTBL). The BCTBL is addressed by 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.

Parameter Definitions

NODSECT — Specification of NODSECT results in the BCTBL fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, the system generates a DSECT with the name BCTBL (plus the optional SUFFIX).

REG — Provides for the optional specification of a register for which a USING statement for the BCTBL fields is generated.

SUFFIX — One ASCII character in length. If provided, all labels are generated by the concatenation of the letters BCTBL, the user provided SUFFIX, and the field name.

Structure

BCTBL		BYTE 0	BYTE 1	BYTE 2	BYTE 3	
BEGIN						
LNBUF	+0	HITCT				
	+4	LOCK1				
REPINUM	+8	MISSCT				
	+C	FILECT	FLAGS	TYPE	SPARE	
	+10	IOHEAD				
	+14	WDATA	WDATAH	WINDEX	WROOT	WTABLE
	+18	WADATA	WAINDEX	WAROOT	WRES	
	+1C	EXPAND				
	+20	BCE1				
	+24					
	+28					
	+2C					
	+30					
	+34					
	+38					
	+3C					
	+40					

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+44				
+48				
+4C				
+50				
+54				
+58	BCE2			
+5C				
+60				
+64				
+68				
+6C				
+70				
+74				
+78				
+7C				
+80				
+84				
+88				
+8C				

Example

```
          BCTBL REG=2,SUFFIX=T
+BCTBLT         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
+*
+BCTBLTBEGIN            DS   OF           (FULLWORD ALIGNMENT)
+*
+*** BUFFER CONTROL TABLE
+*
+BCTBLTNBUF             DS   OHL1         COUNT OF BUFFERS (BCE'S)
+BCTBLTHITCT            DS   A            HIT-COUNT (READ)
+BCTBLTLOCK1            DS   A            BCE LOCK1 (DMS INTERNAL)
+BCTBLTREPLNUM          DS   OHL1         CIRCULAR BCE NUMBER (SCAN)
+* BCTBLHITCT AND BCTBLMISSCT INDICATE PERCENTAGE OF READ OPERATIONS
+* HANDLED WITHIN THE BUFFER POOL      (WITHOUT PHYSICAL IO OPERATION).
+BCTBLTMISSCT           DS   A            MISS-COUNT (READ)
+BCTBLTFILECT           DS   BL1          COUNT OF FILES USING BCT
+BCTBLTFLAGS            DS   BL1          BCTBL FUNCTION FLAGS
```

```
+BCTBLTFLAGSEXT       EQU X'80'        INTERNAL FLAG FOR
+*                                     EXTRACT FUNCTION
+BCTBLTFLAGSRPL       EQU X'40'        GET REPLACEMENT BUFFER
+*                                     WITHOUT IO OPERATION
+BCTBLTTYPE           DS  CL1          BLOCK TYPE FOR FUNCTION
+*                                     (VALUE AS IN BCETYPE)
+BCTBLTSPARE          DS  BL1          SPARE
+BCTBLTIOHEAD         DS  A            HEAD OF CHAIN FOR BCES
+*                                     WITH I/O OUTSTANDING
+BCTBLTWTABLE         DS  XL8          TABLE OF WEIGHTS FOR REPL
+* VALUE IN PAREN BELOW IS DEFAULT     VALUE LOADED BY SVC OPEN.
+                     ORG BCTBLTWTABLE
+BCTBLTWDATA          DS  XL1          DATA BLOCK NO HOLD  (1)
+BCTBLTWDATAH         DS  XL1          DATA BLOCK HOLD     (2)
+BCTBLTWINDEX         DS  XL1          INDEX BLOCK (PRIMARY) (3)
+BCTBLTWROOT          DS  XL1          INDEX ROOT (PRIMARY) (5)
+BCTBLTWADATA         DS  XL1          LOW LEVEL ALT BLOCK (1)
+BCTBLTWAINDEX        DS  XL1          INDEX BLOCK (ALT)   (3)
+BCTBLTWAROOT         DS  XL1          INDEX ROOT (ALT)    (5)
+BCTBLTWRES           DS  XL1          RESERVED WEIGHT CLASS (0)
+BCTBLTEXPAND         DS  BL4          EXPANSION AREA (BCTBL)
+* END OF BCTBL HEADER; BCE'S BEGIN HERE
+BCTBLTBCE1           DS BL56          BUFFER CONTROL ENTRY
+BCTBLTBCE2           DS BL56          BUFFER CONTROL ENTRY 2,ETC
+BEGIN                CODE
+                     USING BCTBLT,2
```

4.2.5 BCTGEN - Generate a Buffer Pool Control Table

Syntax

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

Function

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

Parameter Definitions

NBUF The number of buffers to be included in the buffer pool. The user must supply an absolute expression which evaluates to an integer not greater than 255.

Example

```
 LAB1     BCTGEN NBUF=8
+LAB1     DS     0F
+         DC     AL1(8)        BUFFER COUNT
+         DC     XL31'0'       REMAINDER OF PREFIX
+         DC     (8)XL56'00'   BUFFER CONTROL ENTRY
```

4.2.6 BEGTRANS - DMS/TX Transaction Rollback (SVC 80)

Syntax

```
[label]  BEGTRANS   RETCODE={(register)}[,ACK={YES}][,CANCEL={YES}]
                            {  address }      {NO }         { NO}
```

Function

BEGTRANS marks the beginning of a DMS/TX transaction or subtransaction.

Parameter Definitions

RETCODE   Address where the return code will be stored.

CANCEL   YES specifies to cancel the operation on error detection.

ACK   YES specifies to produce an acknowledge GETPARM when errors are detected.

Return Codes

Code	Definition
0	Success.
4	No recovered files are open.
8	DMX/TX not supported on this system.
12	Invalid function request.
16	Invalid parameter or parameter list.
20	Unable to process before image journal for this task. Run DMSTX utility on this database.
24	Error encountered on this file during rollback. Run DMSTX utility on this file.
28	Specified mark not found. The entire transaction has been rolled back.
32	Unable to set file crash status. File may contain uncommitted updates.
36	Unable to set database crash status. Database may contain uncommitted updates.

Example

```
          BEGTRANS  RETCODE=RCADDR,CANCEL=YES,ACK=NO
+         PUSHA 0,=A(64)
+         MVI   0(15),X'80'           Set last parameter flag
+         PUSHA 0,RCADDR              return code
+         LR    1,15
+         SVC   80 (BEGTRANS)
+         POPN  0,2*4
          END BEGIN
```

4.2.7 CALL - Call a Subroutine

Syntax

```
[label] CALL    EPLOC=address {,PARM={(register)}}
                              { address   }

                              {,PARMLOC=address   }

                [,COND={integer}]
                     {  15   }
```

Function

Provides the necessary linkage to transfer control to another routine. Loads the address of a parameter list (if specified in PARM or PARMLOC) into register R1. Also branches (conditionally) to the label or address specified in EPLOC by means of a JSCI instruction, leaving the return address on the stack. The JSCI instruction

- Saves the contents of control register 1
- Stores general registers 0 to 14 on the stack
- Places the address of the register 0 save area in control register 1, as well as in the stack pointer, (GR 15)

The lowest address in any current static area is, by convention, passed in register R14.

Restrictions

A stack, with stack top addressed by GR 15, must be available to the caller.

Parameter Definitions

EPLOC The address of a word that contains the called routine's entry point. This must be specified in a form allowable in the D2(X2,B2) field of the RX-type assembly instruction format.

PARM The address of a parameter list to be passed in register 1 (R1).

PARMLOC The address of a word that contains the address of a parameter list to be passed in R1 (with format of the address as specified for EPLOC).

COND Specifies the condition code under which the routine is to be called. The default value is 15.

Example

```
 GETPGM  CALL PARM=PADDR,EPLOC=ENTRYWRD,COND=8
+GETPGM  LA      1,PADDR
+        JSCI    8,ENTRYWRD
```

4.2.8 CANCEL - Cancel Program (SVC 16)

Syntax

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

Function

To terminate a program in the event of uncorrectable program failure, such as

- Exhaustion of a system resource
- Illegal or invalid parameters to an SVC routine or other system service program
- A program-detected condition which cannot be satisfactorily resolved within the program

CANCEL causes the transfer of program control to the Help processor for cancellation of the issuing program. The message specified in the MSG parameter, along with a standard CANCEL message, is displayed on the workstation. The user cannot immediately resume program execution by the CONTINUE PROCESSING command processor command. The user may, however, examine the program by means of the Help processor's debugging facilities, modify the current instruction address by means of the Debugger's Inspect and Modify option, and then attempt to resume program execution or issue the CANCEL command to remove the program from the system. A program terminated by a CANCEL supervisor call from within privileged code cannot be continued. The CANCEL macro is used in conjunction with the CEXIT macro.

Restrictions

Must not be issued while in system must complete (SMC) state.

Parameter Definitions

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

Stack On Input

```
                                                Lower
                                                Address
                |__________________|
   0(SP)        |                  |
                | (1) Address of   |            Higher
                |____Message_______|            Address
                |    Preceding     |
                   Stack Data
```

(1) The address of the message to be sent to the user is constructed in the following format:

```
Byte  0_______________4___________10______12_________________N
     | (2)            | (3)        | (4)   | (5)             |
     |_Message number_|_ Issuer ID_|_Length_|_______Text______|
```

(2) Message number in ASCII characters (four bytes). Always required.

(3) Issuer identification in ASCII characters (six bytes).

(4) Length of the message to be sent in binary (two bytes). This is the length of the text which starts at byte 12.

(5) Message text in ASCII characters. If the message is 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.

Stack On Output

The Help processor is entered with no return to issuing program. The user program abnormally terminates when the user issues the CANCEL command in the Help processor.

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'
```

4.2.9 CEXIT - Cancel Exit (SVC 39)

Syntax

Format 1:

```
[label] CEXIT   SET [,([{NODEBUG}][,NOHELP])]
                        {DUMP   }

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

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

Format 2:

```
[label] CEXIT   CANCEL
```

Function

The CEXIT SVC sets or cancels link level parameters which specifiy the way a program handles error conditions.

The SET option allows the issuing program upon detecting an error, to complete one of the following actions:

- bypass the debug processor
- initiate a full program dump
- disable the HELP key
- supply an alternate error handling intercept routine

The CANCEL option negates the effect of any previously issued CEXIT supervisor call in the current link level. Abnormal termination conditions are not intercepted at the current link level and are processed in the normal way by the command processor.

The caller may cancel the parameters specified via the CANCEL option or reset them via another SET option during the same link level or may temporarily override the specified parameters at subsequent link levels by issuing another CEXIT SET at the appropriate link level.

The error handling intercept routine gains control from the cancel processor in the following manner:

- If the abnormal termination condition occurred within the same link level, the cancellation process 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 the entrance to the supervisor call resulting in the abnormal termination condition. A frequent error is to fail to re-establish the addressability in the cancel-exit routine.

- 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 cancellation process does not, in this case, attempt to close any files.

- If the abnormal termination condition occurred within a subsequent link level (that is, a linked-to program, for which no cancellation-interception routine was specified), the cancellation process 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. Control then passes to the cancellation-interception 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 (termination at the present link level or termination at a subsequent level), entry to the cancellation-intercept routine cancels the CEXIT options for the link level. They may be reset by the user via a subsequent CEXIT supervisor call. On the stack, the cancellation-intercept routine may access the following data using the DSECT produced by the CXT macroinstruction:

CANCELLATION PCW	8 bytes
CANCELLED PROGRAM'S NAME	8 bytes
CEXIT OPTION IN EFFECT	1 byte (as in PFBCXTOPTS)
(RESERVED)	47 bytes
GENERAL REGISTERS 0-15	64 bytes
CANCEL MSGLIST	Variable length

Parameter Definitions

NODEBUG The Debugger is bypassed for abnormal termination conditions. Control passes directly to the cancel processor without direct user notification.

DUMP This option also provides a full program dump prior to entry into the cancel processor.

NOHELP — This option causes the HELP key to be disabled at the current link level to enable entry into the Help processor. When the programmer specifies NOHELP, pressing the HELP key in User mode has the following effects:

- If the workstation does not have operator privileges, the alarm is sounded.

- If the workstation is a dual-mode operator console, the system enters Operator mode. This option remains in effect until the user issues a CEXIT without the NOHELP option or until the program unlinks back to either the command processor's initiator or to a link level for which NOHELP was not specified.

Unless specifically disabled within a link level, the NOHELP option is propagated to higher link levels. The NOHELP option should only be utilized in those situations where user access to CANCEL and other system functions must be limited, as in the case of critical sections of application programs updating multiple file chains and pointers. Such programs should be as error-free as possible prior to using this facility.

ADDRESS — Specifies the address of a cancellation-intercept routine provided by the user program.

MESSAGE — Provides text to be used by both the Help processor and the Debugger in place of the Cancel Processing menu descriptions. Specification is of a modifiable data area location containing a 1-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.

Stack On Input

CEXIT SET Option

```
                                         Lower
                                         Address
                |0   1     2     3   |
    0(SP) |     |_____|______________|
                |     |              |
                |(1)  |(2) Address of|
                |FLAG |Intercept RTN |
    4(SP)       |____________________|
                |                    |
                | (3) Address of     |
                |     Help Message   |
    8(SP)       |____________________|
                |                    |
                |     Unused         |   Higher
                |____________________|   Address
                |     Preceding      |
                |    Stack Data      |
(1)  Flags:
        Bit 0     1  =   SET option
        Bits 1-2 00  =   Debug enabled
                 01  =   Nodebug option
                 10  =   PDUMP option
                 11  =   Dump option
        Bit 3     0  =   HELP key enabled
                  1  =   HELP key disabled
```

(2) Address of user error handling intercept routine or zero.

(3) Address of user-supplied PF16 HELP display message to be used in place of the CANCEL PROCESSING default message.

CANCEL Option

```
                                         Lower
            |                      |     Address
            |0_____1______2____3___|
 0(SP) |    | (1) |                |
            |Flags|                |     Higher
            |_____|________________|     Address
            |     Preceding        |
            |    Stack Data        |
```

```
(1)  Flags:
        Bit 0   0   =   CANCEL option
        Remainder of word not examined.
```

Stack On Output

```
                                         Lower
            |                      |     Address
            |______________________|     Higher
 0(SP) |    |     Preceding        |     Address
            |    Stack Data        |
```

Example

```
 EXIT     CEXIT SET,NODEBUG,ADDRESS=FIXPROBS,MESSAGE=CANCELME
+EXIT     PUSHA 0,0
+         MVI   0(15),B'00000000'                X
+                                           PF KEYS 1-8 MASK
+         MVI   1(15),B'00000000'                X
+                                           PF KEYS 9-16 MASK
+         PUSHA 0,CANCELME
+         PUSHA 0,FIXPROBS
+         MVI   0(15),B'10100000'           OPTIONS BYTE
+         SVC   39                          (CEXIT SET)
```

4.2.10 CHARGEN - Macro Processor Large Character Generator

Syntax

```
CHARGEN string
```

Function

This macro generates an 8 x 8 space character for each character input to the macro. It uses the assembly macro instruction MNOTES to generate the string. Therefore, the string prints as a comment in the source listing when the PRINT NOGEN assembly listing control instruction is in effect.

Parameter Definitions

string A character string which cannot continue on another source line. The string can contain embedded blanks if enclosed within single quotes.

Example

```
          CHARGEN WV54LOG
+*
+*   ##   ## ##   ##  ####       #   ##       #####   #####
+*   ##   ## ##   ##  #         ##   ##      ##   ## ##   ##
+*   ##   ## ##   ##  #        # #   ##      ##   ## ##   ##
+*   ##   ## ##   ##  ####    #  #   ##      ##   ## ##
+*   ## # ## ##   ##      #  ######  ##      ##   ## ##  ###
+*   #######  ## ##   #   #      #   ##      ##   ## ##   ##
+*   ### ###   ###    #   #      #   ##      ##   ## ##   ##
+*    #   #     #      ###       #   #######  #####   #####
+*
```

4.2.11 CHECK - Check for Event Occurrence (SVC 17)

Syntax

Format 1:

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

                [,IOSWREG='R0'][,FORM=LIST]
```

Format 2:

```
[label] CHECK   INTERVAL[,FORM=LIST]
```

Format 3:

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

Format 4:

```
[label] CHECK   WSKEY={ address  }[,FORM=LIST]
                      {(register)}
```

Format 5:

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

                [,FORM=LIST]
```

Format 6:

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

                [,FORM=LIST]
```

Format 7:

```
[label] CHECK   SEMA={ address  }[,FORM=LIST]
                     {(register)}
```

Format 8:

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

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

Function

The function of this macro varies slightly depending on the format used.

- Format 1 -- CHECK OFB or VCB waits for completion of an I/O operation. If "intervention required" is indicated on completion, an appropriate workstation message is issued (if possible) to inform the user, and CHECK proceeds when the "intervention required" condition has been cleared. (CHECK may reissue the message if the condition was not corrected.) CHECK waits for completion again after the condition has been cleared. If the operation has not been completed, CHECK suspends processing of the issuing program until it has. In the event of a permanent error completion (IOSW bit EC set, bits NC or IRQ not set), CHECK returns to the address specified by the ERREXIT parameter. Otherwise, CHECK returns to the next sequential instruction address. CHECK logs I/O errors by means of a nonresident subroutine.

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

- Format 3 -- CHECK MESSAGE waits for a message to be sent to the issuing task through the specified port name which the issuing task must have established by making a call to the CREATE SVC.

- Format 4 -- CHECK WSKEY waits for a program function key to be pressed at the specified workstation, which must be reserved for use by the issuing task. An unchecked XIO request must not be outstanding to this workstation when this CHECK is issued. The issuing program is cancelled if an unchecked I/O operation (XIO) has been issued to the specified workstation, or if that workstation is not reserved for use by this task.

- Format 5 -- CHECK INTERRUPT waits for an unsolicited interrupt from a workstation, a printer, or a telecommunications device. For a workstation, this CHECK option waits for a program function key whether the keyboard is locked or not. The issuing task is cancelled if the device is not reserved for use by the issuing task or an unchecked I/O is outstanding. The IOSW of the unsolicted interrupt is moved to the 8-byte area specified in the input parameter list.

- Format 6 -- CHECK TCIO waits for the occurrence of a telecommunications I/O event. This event may be the completion of an I/O operation 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. If the TC I/O is completed with an error because of missing device microcode or missing peripheral processor microcode, the error is logged and the microcode is not loaded.

  Sequencing rules for alternation of the RECEIVE or TRANSMIT macroinstruction followed by CHECK TCIO are enforced by the XIO SVC routine. XIO also checks that the device and the DLP are not exclusively reserved by another task, and that the channel device is currently open (using the IPOPEN SVC).

  CHECK TCIO may be issued without any previous I/O being issued provided the specified device is reserved by the calling task. In this case the CHECK acts as an unsolicited interrupt from the DLP on the specified device. To receive an unsolicited interrupt from the DLP, at least one of the devices on the DLP must have been opened with the IPOPEN SVC 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 is allowed to reissue the CHECK TCIO to receive the IOSW in response to the XIO. The general status byte of the IOSW returned indicates 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 is cancelled if the device is not reserved for use by the issuing task or an unchecked I/O operation is outstanding.

- Format 7 -- CHECK SEMA allows a privileged user to wait upon a supplied semaphore. CHECK SEMA issues a CANCEL if the caller is not privileged.

- Format 8 -- CHECK MULTIPLE waits for any one of several specified events to occur. These events can be any one of the events explained in Formats 1 through 7 above.

Restrictions

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

Parameter Definitions

OFB — For the OFB or VCB 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, 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 parameter is an address pointing to a 4-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 appended to either specification option by the macroinstruction code in order to distinguish the CHECK VCB option from the CHECK OFB option.

ERREXIT — Optional parameter specifying the address of an error handling routine 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 is placed in general registers 0 and 1.

MESSAGE An address in the user modifiable area, where a received message is placed. The receipt area in the user modifiable area must contain the total length of the area, in binary, in its first two bytes. The length must not be greater than 2048 bytes. The message is placed in the specified area. If the area length is less than the message length plus two, the message is 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 4-character name of one of this task's active message receipt ports, as established by CREATE. The port can be specified as an expression that addresses a 4-byte field containing the port name, as a register in parentheses that points 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 is 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. The number 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 the user modifiable area, into which the I/O status word (IOSW) is placed. May be specified as an address expression, or as a register in parentheses containing the address of the IOSW receipt area. This parameter is required for the CHECK INTERRUPT option and for the CHECK TCIO option if the CHECK is for a TC unsolicited interrupt.

The IOSWADDR parameter 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 expression which is the number of events, or as a register in parentheses which contains the number of events (in binary) in the low-order byte.

SEMA — The address of the semaphore upon which to wait. May be specified as an address expression, or as a register in parentheses which contains the address.

FORM — The FORM=LIST parameter may be used with Formats 1-7 above to build a multiple CHECK list on the stack. This example code builds a multiple CHECK list which waits for a semaphore (SIGNAL), a PF key, or a timer, in that order:

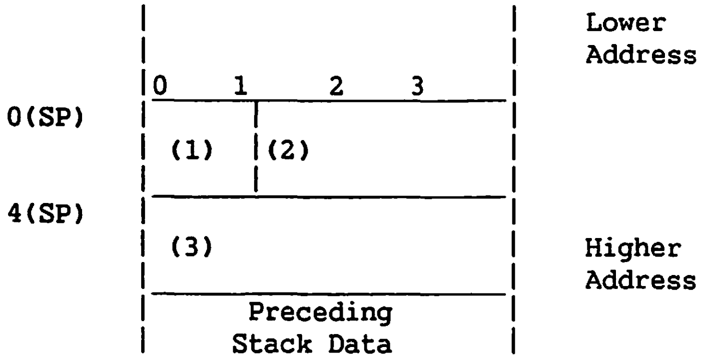
```
CHECK INTERVAL,FORM=LIST
CHECK WSKEY=(R1),FORM=LIST
CHECK SEMA=SIGNAL,FORM=LIST
LR    R9,SP
CHECK MULTIPLE,PLIST=(R9),COUNT=3
```

After the call to CHECK MULTIPLE, the top stack word contains the offset into the parameter list of the event that occurred. The parameter list remains on the stack.

Stack On Input

Single Event CHECK

For a single-event CHECK, a single 8-byte data structure as described under multiple event check is put on top of the stack.



Multiple Event CHECK

```
            |                       |   Lower
            |                       |   Address
            |0    1    2    3       |
 0(SP)      |     |                 |
            | (1) | (2) Address of  |
            |_____| Parameter List  |
 4(SP)      |                       |
            | (3) Count of items    |   Higher
            |_____being checked_____|   Address
            |      Preceding        |
            |     Stack Data        |
```

(1) Count of the number of parameter items contained in the parameter list (bits 1-7). Bit 0 of this byte is set to 1.

(2) The address of a parameter list of items constructed as shown below.

(3) Count of the number of items being checked.

For multiple-event CHECK, if the option flag (byte 0) on the input parameter list for the event is set 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 should be used to build a multiple-event CHECK list on the stack. See CHECK macroinstruction description for further detail.

Each 8-byte data structure is constructed as follows:

```
                |                       |   Higher
                |____ARGUMENT LIST______|   Address
                |0    1    2    3       |
PLIST ADR       |     |                 |
                | (a) | (b)             |
                |_____|_________________|
                |                       |
                | (c)                   |
                |_______________________|   Lower
                |                       |   Address
```

(1) Normal I/O check (OFB) item:
- (a) Byte 0: zero
- (b) Bytes 1-3: OFB address
- (c) 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 return the completion IOSW in general registers 0 and 1.

(2) VOLIO I/O check (VCB) item:

(a) Byte 0: zero

(b) Bytes 1-3: VCB address plus 1

(c) 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 return the completion IOSW in general registers 0 and 1.

(3) Timer check item:

(a) Byte 0: X'10'

(b) Bytes 1-7: ignored

(4) Intertask message check item:

(a) Byte 0: X'20'

(b) Bytes 1-3: address of an area in the user modifiable area where a message is received. The first two bytes of this area must contain its length in bytes (binary) including these bytes. This length must not be greater than 2048. The message (not including its length bytes) is moved to the area following these bytes, truncated if it is too long for the specified area. The area's length bytes are adjusted to reflect the length of the message, including these bytes.

(c) Bytes 4-7: the name (CL4) of one of this task's active ports, as established by the CREATE SVC.

(5) Workstation program function key check item:

(a) Byte 0: X'40'

(b) Bytes 1-3: workstation device number

(c) Bytes 4-7: ignored

(6) Unsolicited interrupt item:

(a) Byte 0: X'08'

(b) Bytes 1-3: number of any device on line

(c) Bytes 4-7: address of 8-byte area to receive IOSW (must be in user-modifiable-area buffer area or in stack as validated by MCBRWTST)

(7) TC event item:

(a) Byte 0: X'01'

(b) Bytes 1-3: the OFB address of the TC device on which XIO was issued

(c) Bytes 4-7: the address of an 8-byte receiving area for the completion IOSW, or binary zeroes if the IOSW is not desired

(8) Semaphore check item:

(a) Byte 0: X'02'

(b) Bytes 1-3: the address of the semaphore upon which to wait

(c) Bytes 4-7: reserved; must be zero

Stack On Output

Single-event CHECK

Inputs popped from stack. For I/O completion CHECK, a workstation message is displayed, if possible, on conditions that require intervention. See the XMIT SVC description for the format of the 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 event which has occurred. Device intervention required conditions must be handled by the CHECK issuer, who must reissue a CHECK (single- or multiple-event) to wait for I/O completion.

```
                                            Lower
               |                      |     Address
               |______________________|
   0(SP)       |                      |
               | (1) Displacement     |     Higher
               |_____into Event List__|     Address
               |      Preceding       |
               |     Stack Data       |
```

(1) Displacement into the parameter list of the item corresponding to the event that occurred. The displacement value starts from 0, and is incremented in multiples of eight.

Examples

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


 TIMR     CHECK INTERVAL
+TIMR     PUSHN 0,8
+         MVI   0(15),X'10' CHECK INTERVAL
+         SVC   17 (CHECK)
```

```
          CHECK SEMA=(R4)
+         PUSHA 0,0               RESERVED - MUST BE ZERO
+         PUSH  0,R4              PUSH SEMAPHORE ADDRESS
+         MVI   0(15),X'02'       INDICATE SEMAPHORE CHECK
+         SVC   17 (CHECK)
```

4.2.12 CLOSE - Close File (SVC 1)

Syntax

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

Function

Closes a file. CLOSE places the user file block (UFB) in a state in which the OPEN SVC can return the file to processable status. This includes placing sufficient file location information in the UFB so that a succeeding OPEN refers to the same file, volume, and device. If the UFB bit UFBF1WORK is set and the file is in a library named #xxxWORK (where xxx is the USERID), the file is deleted as well as closed. The UFB address is pushed onto the stack before the system issues the CLOSE SVC. On return, the UFB address is removed from the stack.

Parameter Definitions

UFB — The address of a user file block of 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 — For magnetic tape volumes only. If specified, the file is not closed, but rather is positioned so that the first record on the next volume (if any) is provided on the next READ, or written on the next WRITE.

NOREWIND — For magnetic tape volumes only. If specified, rewinding is suppressed when the file is closed.

UNLOAD — For magnetic tape volumes only. If specified, the tape is rewound, set to offline, and effectively dismounted when the file is closed.

Examples

```
LAB1     CLOSE UFB=(R3)
+LAB1    PUSH  0,R3
+        MVI   0(15),B'00000000'    FLAGS
+        SVC   1 (CLOSE)


         CLOSE REEL,UFB=(R2)
+        PUSH  0,R2
+        MVI   0(15),B'10000000'    FLAGS
+        SVC   1  (CLOSE)
```

```
 END      CLOSE NOREWIND,UFB=(R2)
+END      PUSH  0,R2
+         MVI   0(15),B'01000000'    FLAGS
+         SVC   1  (CLOSE)
```

4.2.13 COMMIT - Commit Resources (SVC 52)

Syntax

```
[label] COMMIT  [ALL={YES}][,LEVELS={(Register)}]
                     {NO }         {expression}

                [CANCEL={YES}]  [,ACK={YES}]
                        {NO }         {NO }
```

Function

Commits resources for the user through the sharing task including files, records and extension rights.

Parameter Definitions

ALL — Indicates that all transaction levels should be committed.

LEVELS — Number of levels to be committed. The value can be an expression or a register in parentheses. If it is a register, the register must contain the address of a fullword which gives the level number.

CANCEL — Indicates whether to cancel on errors.

ACK — Indicates whether to issue an acknowledge GETPARM on errors.

Stack On Input

```
             |                       |    Lower
             |                       |    Address
             |0     1      2    3____|
     0(SP)   |_____________________  |
             |      |      |         |
             | (1)  | (2)  |(3)      |
             |______|______|_________|
     4(SP)   |                       |
             |                       |    Higher
             |_______________________|    Address
             |       Preceding       |
             |      Stack Data       |
```

(1) Flag byte:

Bit 0	1 = HOLD, 0 = RELEASE
Bit 1	1 = EXTENSION RIGHTS request
Bit 2	1 = RELEASE ALL and commit DMS/TX transaction
Bit 3	1 = TIME OUT in use
Bit 4	1 = Cancel on error
Bit 5	1 = Produce ACK GETPARM on error
Bit 6	Reserved for internal use, must be zero
Bit 7	Reserved for internal use, must be zero

(2) Time out value in seconds from 0 - 255

(3) Reserved, must be 0

Stack On Output

```
             |                     |   Lower
             |                     |   Address
             |0    1     2    3    |
      0(SP)  |---------------------|
             |  (1) Return Code    |
             |                     |
      4(SP)  |---------------------|
             | (2) User ID|  (3)   |   Higher
             |            |        |   Address
             |---------------------|
             |     Preceding       |
             |    Stack Data       |
```

(1) Return code
(2) User ID of user holding extension rights
(3) Unused

Output

Return Codes

Code	Definition
0	Success.
4	Timeout.
8	Invalid function sequence.
12	Request to HOLD or FREE with no shared files open.
16	System error: the sharer is not active or has run out of memory space.
20	System error: before image journal error during an end transaction.
24	Invalid function parameter.

Example

```
          COMMIT ALL=NO,LEVELS=2,CANCEL=YES,ACK=YES
+         PUSHA 0,2                Push LEVELS parameter
+         PUSHA 0,0
+         MVI   0(15),X'20'         FREE ALL SHARED RESOURCES
+         OI    0(15),X'08'         SET UP CANCEL-ON-ERROR CONDITION
+         OI    0(15),X'04'         SET UP ACKNOWLEDGE-ERROR CONDITION
+         SVC   52 (FREEALL)
```

4.2.14 CREATE - Create Intertask Message Port (SVC 37)

Syntax

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

                 RESIDENT={YES}  [,PRIVILEGED]
                          {NO }

             KEEP={YES}
                  {NO}
```

Function

Allows the issuing task to receive intertask messages sent by the XMIT SVC by establishing a buffer called a message port. When a message is sent to a task, the message is copied from the system message buffer to this message receipt port. If the specified message receipt port was created with the PRIVILEGED option, the SVC rejects any messages generated from nonprivileged state code or from tasks that are not dedicated system tasks.

Creates a resident or nonresident message port with the specified port name and the issuing task as the valid receiver. The port enables a task to accept messages from other tasks. CREATE optionally screens out messages not transmitted by privileged code or by dedicated system tasks.

After creating the message receipt port, a task would use the XMIT macro to transmit messages and the MESSAGE function of the CHECK macro to wait for the receipt of messages in the specified port.

All message receipt ports are destroyed during an UNLINK to free space. To maintain message receipt ports, the KEEP parameter must be specified as YES. Then, message receipt ports are maintained until return to the command processor (link level 0). The default is NO.

Restrictions

Only dedicated system tasks may use the parameter RESIDENT = YES.

Parameter Definitions

PORT The 4-character name of a message receipt port (chosen by the issuing program; any characters are allowed). The name can be specified as a register in parentheses pointing to the port name, as a character string 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. The space can not be greater than 2048 bytes.

RESIDENT Specifying YES makes the message port memory resident at all times. If NO is specified, the message port may be paged in and out of memory as necessary.

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

KEEP When KEEP=YES, the message receipt ports created will not be destroyed until return to the command processor (link level 0).

Stack On Input

```
              |                      |   Lower
              |                      |   Address
              |0     1    2    3     |
      0(SP)   |______|____|_(2)______|
              | (1)  |    | Buffer   |
              |      |    | Size     |
      4(SP)   |______|____|__________|
              | (3)  Message Receipt |   Higher
              |      Port Name       |   Address
              |______________________|
              |      Preceding       |
              |     Stack Data       |
```

(1) Flags
    X'80' - Sets XMBUFFLAGPRSYS which limits receipt of messages to those sent by privileged code and dedicated system tasks.
    X'01' - Sets XMBUFFLAGRS so that the message receipt port is always memory resident.

(2) Buffer size: Size of the buffer to allocate for messages. Cannot be greater than 2048.

(3) Message receipt port name: a 4-character name to be assigned to the message receipt port.

Stack On Output

```
              |                      |   Lower
              |                      |   Address
              |______________________|
      0(SP)   |                      |
              |     Return Code      |   Higher
              |______________________|   Address
              |      Preceding       |
              |     Stack Data       |
```

Output

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

Code	Definition
0	Success.
4	Another task has activated the specified port name.
8	Same task has already activated the specified port name.
12	GETMEM failure.

Example

```
 CMSG    CREATE PORT=PORTNAME,BUFSIZE=(R0)
+CMSG    PUSHC 0(4,0),PORTNAME                  PORT NAME
+        PUSH  0,R0                             BUFFER SIZE
+        MVI   0(15),X'00'
+*
+        SVC   37 (CREATE)
```

4.2.15 CXT - 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-intercept routine.

Parameter Definitions

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

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

SUFFIX One ASCII character in length. If provided, all labels are generated by the concatenation of the characters CXT, the user-provided SUFFIX, and the field name.

Structure

CXT BEGIN

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+0	PCW			
+4				
+8	PROGRAM			
+C				
+10	FLAGS	SPARE		
+14				
+18				
+1C				
+20				
+24				
+28				
+2C				
+30				
+34				
+38				
+3C				

		BYTE 0	BYTE 1	BYTE 2	BYTE 3
	+40	REGS			
	+44				
	+48				
	+4C				
	+50				
	+54				
	+58				
	+6C				
	+60				
	+64				
	+68				
	+6C				
	+70				
	+74				
	+78				
	+7C				
MSGLIST	+80	MSGID			
	+84	MSGISSUER			
	+88			MSGLENGTH	

Example

```
 CXT      CXT
+CXT        DSECT
+*        THE CEXIT RETURN INFORMATION BLOCK IS RETURNED TO
+*        A PROGRAM'S CEXIT ROUTINE FOR PROGRAMMED ANALYSIS
+*        OF AN ABNORMAL TERMINATION CONDITION.
+*        DATE 03/28/79
+*        VERSION 4.00
+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       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
```

4.2.16 DELETE - Delete Record From Indexed File

Syntax

```
[label] DELETE {EOF},UFB={(register)}[,COND={     integer       }]
               {REL}     { address  }       {absolute expression}
                                            {        15         }
```

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.

Restrictions

The file specified must be opened in record access method (RAM) for I/O or Shared mode processing. The last function on this file must have been a successful READ with the HOLD option. In Shared mode, the lock on the record to be rewritten must not have been released by an intervening operation on any other shared file.

Parameter Definitions

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 address 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 parameter 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 under any condition.

EOF   Deletes records following the present record to the end of the file. This option is used for relative files only.

REL   Deletes a record for relative file organization only. Performing a READ HOLD is not required before the delete.

Output

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
- Invalid key (DELETE REL) -- UFBFS1 = 2, UFBFS2 = 3

NOTE

Register 1 is loaded with the address of the UFB.

Examples

```
 LAB2    DELETE UFB=(R2)
+LAB2    LR     1,R2                   SET REGISTER 1
+        JSCI   15,12(1)               DELETE FUNCTION


 LAB3    DELETE UFB=DSKUFB,COND=7
+LAB3    LA     1,DSKUFB               SET REGISTER 1
+        JSCI   7,12(1                 DELETE FUNCTION
```

4.2.17 DESTROY - Destroy Intertask Message Receipt Port (SVC 38)

Syntax

```
[label] DESTROY PORT={(register)}
                     { address  }
                     { 'string' }
```

Function

Deallocates the intertask message receipt port with the specified port name. The port must have been activated by the same task using the CREATE macroinstruction.

Parameter Definitions

PORT The 4-character name of a message receipt port. The name can be specified as a register in parentheses which points to the port name, as a character string in single quotes which is the port name, or as an expression addressing a 4-byte field which contains the port name.

The issuing task's named message receipt port is deallocated. The buffer space is returned to the appropriate memory pool and the port name is released for further use.

Stack On Input

```
        |                      |   Lower
        |                      |   Address
        |______________________|
 0(SP)  |                      |
        |   Message Receipt    |   Higher
        |______Port Name_______|   Address
        |      Preceding       |
        |     Stack Data       |
```

Stack On Output

```
        |                      |   Lower
        |                      |   Address
        |______________________|
 0(SP)  |                      |
        |  Return Code         |   Higher
        |______________________|   Address
        |      Preceding       |
        |     Stack Data       |
```

Output

A return code is placed in the topword of the stack top as follows:

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

Example

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

4.2.18 DEXIT - DMS/TX Deadlock Exit (SVC 81)

Syntax

```
    DEXIT    {  SET    }[,CANCEL={NO }][,ACK={NO }][,RETCODE={(register)}]
             { CLEAR   }         {YES}       {YES}           {  address }
             {CLEARALL}

                        [,STATUS={(register)}][,ADDRESS={(register)}]
                                 {  address }           {  address }
```

Function

The DEXIT SVC is used to establish an address to which control is returned after deadlock processing when a deadlock is detected under DMS/TX. This SVC sets and clears deadlock exits. There is a maximum limit of 24 deadlock exits per link level. The parameters for a deadlock exit are the address of the deadlock exit, and the address of a fullword which receives the ROLLBACK return code. If the status code address is zero, any errors encountered during deadlock processing cause the program to cancel. The SUSPEND function suspends the current DEXIT, so the exit is not taken. There are two options for clearing a deadlock exit. The first is to clear the most recently established deadlock exit at the current link level. Clearing a suspended DEXIT results in its reactivation. The second option is to clear all deadlock exits for this link level. See the VS DMS/TX Reference for more information.

Parameter Definitions

SET Function request -- set a DEXIT.

CLEAR Function request -- clear the most recent DEXIT.

CLEARALL Function request -- clear all DEXITs at this link level.

ACK Specifies production of acknowledge GETPARM for errors.

CANCEL Specifies cancellation on errors.

RETCODE Address at which to store the return code on exit from the DEXIT SVC.

ADDRESS Address to return to after deadlock processing; meaningful only for SET function.

STATUS Address at which to place deadlock ROLLBACK and FREEALL return codes. This parameter is optional; it is meaningful only for SET function.

Stack on Input

Register 1 points to a 4-word parameter list constructed as follows:

```
 |                      |    Lower
 |                      |    Address
 |______________________|
 |  (1)                 |
 |  Address of          |
 |__Return Code_________|
 |  (2)                 |
 |  Address of          |
 |__Function Request____|
 |  (3)                 |
 |  Address of          |
 |__Exit Address________|
 |  (4)                 |
 |  Address of          |    Higher
 |__Status Word_________|    Address
 |                      |
 |                      |
```

(1) Address of the memory location where the return code is to be stored.

(2) Address of function request -- one word constructed as follows:
    Byte 0 - error handling
        Bit 0: 1 = Cancel on error. If the cancel flag is set, then the ACK flag is ignored and the user is cancelled on error.
        Bit 1: 1 = Issue acknowledge GETPARM on error and produce a return code.
        Bits 2-7: Reserved, must be 0.

    Bytes 1-2 -- Reserved, must be 0.
    Byte 3 -- Function request code:
        0 = Set deadlock exit.
        1 = Clear deadlock exit.
        2 = Clear all deadlock exits in this link level.
        3 = Suspend current DEXIT.
        4 = Reactivate current DEXIT.

(3) Address where the deadlock exit address is stored.

(4) Address of the fullword which receives the deadlock exit status. The first halfword contains ROLLBACK return code and the second halfword contains FREEALL return code.

Stack on Output

The return code from the deadlock exit SVC is stored at the address supplied on input to the SVC.

Output

A return code is placed in the topword of the stack.

Return Codes

Code	Description
0	Success.
4	Invalid function request parameter.
8	Invalid ROLLBACK and FREEALL return code address parameter.
12	No deadlock exit found for this link level.
16	Cannot set more than 24 deadlock exits per link level. No exit set.
20	GETMEM failure while trying to set deadlock exit.
28	Invalid parameter list or parameter address.
32	Invalid deadlock exit address.
36	DMS/TX not supported on system. No deadlock exit set.

Examples

```
            DEXIT SET,STATUS=(R2),ADDRESS=DLKADR,RETCODE=(R3)
+           PUSH  0,R2
+           MVI   0(15),X'80'         SET 'LAST' PARAMETER BIT
+           PUSHA 0,DLKADR
+           PUSHA 0,=A(0)             PUSH FUNCTION PARAMETER
+           PUSH  0,R3                PUSH POINTER TO RETCODE
+           LR    1,15                POINT R1 TO PARAMETER LIST
+           SVC   81 (DEXIT)
+           POPN  0,4*4
```

```
            DEXIT CLEAR,RETCODE=(R2)
+           PUSHA 0,=A(1)             PUSH FUNCTION PARAMETER
+           MVI   0(15),X'80'         SET 'LAST' PARAMETER BIT 01
+           PUSH  0,R2                PUSH POINTER TO RETCODE
+           LR    1,15                POINT R1 TO PARAMETER LIST
+           SVC   81 (DEXIT)
+           POPN  0,2*4

            DEXIT CLEARALL,CANCEL=YES,RETCODE=(R2)
+           PUSHA 0,=A(-2147483646)        PUSH FUNCTION PARAMETER
+           MVI   0(15),X'80'         SET 'LAST' PARAMETER BIT 01
+           PUSH  0,R2                PUSH POINTER TO RETCODE
+           LR    1,15                     POINT R1 TO PARAMETER LIST
+           SVC   81 (DEXIT)
+           POPN  0,2*4
```

4.2.19 DFB - Describe Document File Block

Syntax

```
DFB NODSECT[,SUFFIX=character]
```

Function

Describes the structures utilized to pass parameters to and from the Wang VS Document Access subroutines. A copy of the Document File Block (DFB) should be brought into the user's assembly program via the CALL macro. It may be used to generate a DSECT or a data area within a program's static area. Multiple such DSECTS and/or data areas may be created via repeated use of this macro with the SUFFIX option.

Parameter Definitions

NODSECT   Specification of NODSECT results in the DFB fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, the system generates a DSECT with the name DFB (plus the optional suffix).

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

Example

```
         DFB SUFFIX=A
+***********************************************************************
+***********************************************************************
+**                                                                   **
+**      "DFB"  -  Document File Block                                **
+**                                                                   **
+**      Version 2.00.04  -  July 6, 1982                             **
+**      Wang VS Document Access Subroutines                          **
+**                                                                   **
+**      These structures are utilized to pass parameters to and     **
+**      from the Wang VS Document Access Subroutines.  A copy of     **
+**      the DFB should be brought into the user's Assembly           **
+**      Program via Macro Call.  It may be used to generate a        **
+**      DSECT or a data area within a program's static area.        **
+**      Multiple such DSECTs and/or data areas may be created        **
+**      via repeated use of this macro with the &SUFFIX= option.    **
+**                                                                   **
+***********************************************************************
+***********************************************************************
+***********************************************************************
```

```
+DFBA                DSECT
+DFBAFULLDOCID       DS   OCL24     These fields are utilized to    *
+                                   identify the document to be     *
+                                   processed by the user's program.
+*                                  They must be filled in
+*                                  as indicated in each field
+*                                  prior to calling WPOPEN.  They
+*                                  may not be modified until
+*                                  WPCLOSE is called.
+DFBADOCUMENTID      DC   CL4' '    Must have a 4-digit number as   *
+                                   value unless document is being  *
+                                   created for the first time in
+*                                  which case spaces indicates that
+*                                  the document is to be numbered
+*                                  as the next document in the
+*                                  library.
+DFBASPARE01         DC   CL4' '    [Reserved]
+DFBALIBRARY         DC   CL1' '    Document library must always    *
+                                   be provided.
+DFBASPARE02         DC   CL7' '    [Reserved]
+DFBAVOLUME          DC   CL6' '    Volume name normally should be  *
+                                   filled with spaces in which     *
+                                   case the document library's
+*                                  default volume is automatically
+*                                  provided.  (WPOPEN will provide
+*                                  that volume name upon return.)
+*                                  Any program-supplied value in
+*                                  this field overrides the
+*                                  document library's default
+*                                  volume value.
+DFBAVOLUMEDFLT      EQU  C' '
+DFBASPARE03         DC   CL2' '    [Reserved]
+DFBASPARE03         DC   CL2' '    [Reserved]
+DFBARETURNSTATUS    DS   OCL12     These fields are used to
+                                   communicate the results of the
+                                   most recent function call back
+*                                  to the calling program.  Common
+*                                  values are used by all the
+*                                  functions - a value of 0 always
+*                                  indicates unqualified success.
+DFBARETURNCODE      DC   H'0'      Contains the primary status
+                                   should be tested first.  If it
+                                   indicates a DMS error, the
+*                                  DFBFILESTATUS1 must be examined
+*                                  for further information.
+DFBARCSUCCESS       EQU  0         Unqualified success.
+                                   See VS Document Access
+                                   Subroutines Documentation for
+*                                  significance of other return
+*                                  codes.
```

```
+DFBARCPAGELIMIT        EQU 3
+DFBARCTXTOVERFLW       EQU 4
+DFBARCTXTNOTFND        EQU 5
+DFBARCHEADER           EQU 6
+DFBARCPRTPRM           EQU 7
+DFBARCENDFORMAT        EQU 8
+DFBARCENDSCTION        EQU 9
+DFBARCEMPTYLIBR        EQU 10
+DFBARCLOW              EQU 1100
+DFBARCHIGH             EQU 1186
+DFBARCDOCUMENTID       EQU 1101
+DFBARCLIBRARY          EQU 1102
+DFBARCVOLUME           EQU 1103
+DFBARCOPENMODE         EQU 1111
+DFBARCPROTECTCL        EQU 1112
+DFBARCRECORDCNT        EQU 1113
+DFBARCRETENTION        EQU 1114
+DFBARCPAGE             EQU 1121
+DFBARCELEMENT          EQU 1122
+DFBARCCHARACTER        EQU 1123
+DFBARCLENGTHREQ        EQU 1124
+DFBARCUNITACCESS       EQU 1125
+DFBARCMODEACCESS       EQU 1126
+DFBARCSEARCHMODE       EQU 1128
+DFBARCNAME             EQU 1131
+DFBARCOPERATOR         EQU 1132
+DFBARCAUTHOR           EQU 1133
+DFBARCCOMMENTS         EQU 1134
+DFBARCDATECR           EQU 1135
+DFBARCTIMECR           EQU 1136
+DFBARCWORKTIMECR       EQU 1137
+DFBARCKEYSCR           EQU 1138
+DFBARCDATERE           EQU 1139
+DFBARCTIMERE           EQU 1140
+DFBARCWORKTIMERE       EQU 1141
+DFBARCKEYSRE           EQU 1142
+DFBARCWORKTIMETO       EQU 1143
+DFBARCKEYSTOTAL        EQU 1144
+DFBARCLINESTOTAL       EQU 1145
+DFBARCLINESPAGE        EQU 1147
+DFBARCDATEPR           EQU 1148
+DFBARCTIMEPR           EQU 1149
+DFBARCDATEAR           EQU 1150
+DFBARCTIMEAR           EQU 1151
+DFBARCARCHIVEID        EQU 1152
+DFBARCPASSWORD         EQU 1153
+DFBARCFROMPAGE         EQU 1161
+DFBARCTHRUPAGE         EQU 1162
+DFBARCSTARTPAGE        EQU 1163
+DFBARC1STHEADERP       EQU 1164
```

```
+DFBARC1STFOOTERP       EQU 1165
+DFBARC1STFOOTERL       EQU 1166
+DFBARCPAPERLNGTH       EQU 1167
+DFBARCMARGIN1          EQU 1168
+DFBARCMARGIN2          EQU 1169
+DFBARCCOPYCOUNT        EQU 1170
+DFBARCFORMNUMBER       EQU 1171
+DFBARCCHARACSET1       EQU 1172
+DFBARCCHARACSET2       EQU 1173
+DFBARCPRINTAREA        EQU 1174
+DFBARCPRINTCLASS       EQU 1175
+DFBARCPRINTTYPE        EQU 1176
+DFBARCDISPOSITN        EQU 1177
+DFBARCFORMSTYPE        EQU 1178
+DFBARCHORIZONTAL       EQU 1179
+DFBARCFINALDRAFT       EQU 1180
+DFBARCFORMAT           EQU 1181
+DFBARCDOCSUMMARY       EQU 1182
+DFBARCVERTICAL         EQU 1183
+DFBARCDOCUMENTI2       EQU 1184
+DFBARCLIBRARY2         EQU 1185
+DFBARCVOLUME2          EQU 1186
+DFBARCBADTEXT          EQU 1200
+DFBARCBADFORMAT        EQU 1201
+DFBARCDMS              EQU 2100
+DFBARCDMS2ND           EQU 2200
+DFBARCDMSPROTO         EQU 2300
+DFBARCDMSMAP           EQU 2400
+DFBARCDMSQUEUE         EQU 2500
+DFBARCNOTOPEN          EQU 3001
+DFBARCWRONGMODE        EQU 3002
+DFBARCBEYONDPAGE       EQU 3003
+DFBARCDOCFULL          EQU 3004
+DFBARCINUSE            EQU 3005
+DFBARCALREADYOPN       EQU 3006
+DFBARCDOCEXISTS        EQU 3007
+DFBARCNODOCUMENT       EQU 3011
+DFBARCREVISION         EQU 3013
+DFBARCNOTFOUND         EQU 3014
+DFBARCDAMAGED          EQU 3015
+DFBARCADDRESSSP        EQU 3017
+DFBARCNOTHELD          EQU 3018
+DFBARCOVERQUEUE        EQU 3020
+DFBARCLIBRFULL         EQU 3021
+DFBARCNOFORMAT         EQU 3022
+DFBARCPRVTOCERR        EQU 3023
+DFBARCPRXMITERR        EQU 3024
+DFBARCVOLNOMOUNT       EQU 3025
+DFBARCVOLEXCLUSV       EQU 3026
+DFBARCVOLVTOCERR       EQU 3027
```

```
+DFBARCNOBUFFERS        EQU 3028
+DFBARCSTRINGPARM       EQU 4000
+DFBARCPARAMOVRLP       EQU 4001
+DFBAFILESTATUS1        DC  X'00'      Significant only if a DMS error
+                                      is indicated above and can be
+                                      used to determine whether
+*                                     subsequent information is
+*                                     provided in either
+*                                     DFBFILESTATUS2 or DFBFILECANCEL.
+DFBAFS1SUCCESS         EQU C'0'       Values as per UFBFS1.
+DFBAFS1ATEND           EQU C'1'
+DFBAFS1INVKEY          EQU C'2'
+DFBAFS1IOERR           EQU C'3'
+DFBAFS1CANCEL          EQU C'6'
+DFBAFS1TIME            EQU C'7'
+DFBAFS1SHARE           EQU C'8'
+DFBAFS1OTHER           EQU C'9'
+DFBAFILESTATUS2        DC  X'00'      Significant only if a DMS error
+                                      indicated by DFBRETURNCODE and
+                                      DFBFILESTATUS1 does not indicate
+*                                     that a cancel code has been
+*                                     stored in DFBFILECANCEL.
+*                                     For values that indicate format
+*                                     or conflict errors, check the
+*                                     value DFBFILESTATUSX.
+DFBAFS2NOINFO          EQU C'0'       Values as per UFBFS2.
+DFBAFS2BYVIOL          EQU C'4'
+DFBAFS2ACC             EQU C'5'
+DFBAFS2RESERR          EQU C'6'
+DFBAFS2INVFUN          EQU C'5'
+DFBAFS2XFILE           EQU X'80'
+DFBAFS2XLIB            EQU X'40'
+DFBAFS2XVOL            EQU X'20'
+DFBAFS2XSPACE          EQU X'10'
+DFBAFS2XVTOC           EQU X'08'
+DFBAFS2XPOS            EQU X'04'
+DFBAFS2XPROT           EQU X'02'
+DFBAFS2XFORMAT         EQU X'01'
+DFBAFILESTATUSX        DC  X'00'      Significant only if a DMS error
+                                      is indicated by DFBRETURNCODE
+                                      and DFBFILESTATUS2 indicates
+*                                     a format or conflict error.
+DFBAFSXNOINFO          EQU X'00'      Values as per UFBXCODE.
+DFBAFSXUSE             EQU X'01'
+DFBAFSXDET             EQU X'02'
+DFBAFSXVOLX            EQU X'03'
+DFBAFSXPOSS            EQU X'04'
+DFBAFSXAOPEN           EQU X'07'
+DFBAFSXAUSE            EQU X'08'
+DFBAFSXDNWP            EQU X'14'
+DFBASPARE04            DC  XL3'00'    [Reserved]
```

```
+DFBAFILECANCEL         DC  XL4'00'     Significant only if a DMS error
+                                       is indicated by DFB-RETURN-CODE
+                                       and DFBFILESTATUS1 indicates
+*                                      that a cancel code has been
+*                                      stored.  (This normally
+*                                      indicates an unusual condition
+*                                      for which an "Open Exit" cannot
+*                                      be used to intercept an error.
+DFBAOPENINFO           DS  0CL16       These fields are used to
+                                       communicate certain parameters
+                                       to and from WPOPEN.  They must
+*                                      be filled in as indicated prior
+*                                      to calling WPOPEN (or WPCHMODE).
+*                                      Subsequent program modifications
+*                                      are ignored by other functions.
+DFBAOPENMODE           DC  CL1'I'      Indicates the mode by which the
+                                       document is processed by
+                                       the calling program:
+DFBAOPENINPUT          EQU C'I'        Input
+DFBAOPENUPDATE         EQU C'U'        Update
+DFBAOPENEXTEND         EQU C'E'        Extend
+DFBAOPENOUTPUT         EQU C'O'        Output
+DFBAOPENDEFAULT        EQU C'D'        Default Output
+DFBAPROTECTCLASS       DC  XL1'00'     File Protection Class - needs
+                                       only be specified for "O" output
+                                       and "D" default output modes.
+DFBAPROTECTDFLT        EQU X'00'       Default to user's default file
+                                       protection class.
+DFBAPROTECTNONE        EQU C' '        No protection to be applied to
+                                       document.
+DFBARECORDCOUNT        DC PL4'0'       Record count - needs only be
+                                       specified for "O" output and
+                                       "D" default output modes.  A zero
+*                                      value indicates that the system
+*                                      should estimate a record count
+*                                      to accommodate a document of
+*                                      several pages.
+DFBARETENTION          DC PL2'0'       Retention period (days) - needs
+                                       only be specified for "O" output
+                                       and "D" default output modes.  A
+*                                      zero value indicates that the
+*                                      document file may be deleted at
+*                                      any time.  Any other value, up
+*                                      to 999 indicates a number of
+*                                      days in the future from which
+*                                      an "expiration date" may be
+*                                      calculated by the system.
+DFBASPARE05            DC XL8'00'      [Reserved]
+DFBAACCESSPARAMS       DS 0XL40        These fields are used to
+                                       communicate additional
+                                       parameters to one or more of the
+*                                      document access subroutines.
```

```
+DFBALOCATION           DS 0XL16       These variables communicate the
+                                      desired starting location for
+                                      certain functions.  Upon return
+*                                     from most functions, these
+*                                     fields are updated with the
+*                                     "current" location.
+DFBAPAGE               DC PL4'0'      "Page" number (-2 to 120)
+DFBAELEMENT            DC PL4'0'      "Element" within "Page"
+DFBACHARACTER          DC PL4'0'      "Character" number within
+                                      "Element" within "Page"
+DFBASPARE06            DC XL4'00'     [Reserved]
+DFBATEXTLENGTHS        DS 0XL8        These variables communicate
+                                      data lengths to and from
+                                      functions that have second
+*                                     arguments that are variable
+*                                     length text strings.
+DFBALNGTHREQ           DC H'0'        Length of data submitted to or
+                                      maximum data length to be
+                                      received from function.
+DFBALNGTHACTUAL        DC H'0'        Indicates actual count of bytes
+                                      of data transferred by function.
+DFBASPARE07            DC XL4'00'     [Reserved]
+DFBAUNITACCESS         DC C'P'        Used to specify the maximum
+                                      extent of WPREAD and WPSEARCH
+                                      operations.
+DFBAUNITDOCUMENT       EQU C'D'       To end of document
+                                      (WPSEARCH only)
+DFBAUNITPAGE           EQU C'P'       To end of current page
+DFBAUNITELEMENT        EQU C'E'       To end of current text element
+DFBAMODEACCESS         DC C'S'        Used to specify Sequential
+                                      or Random mode for WPREAD and
+                                      WPWRITE operations.
+DFBAMODESEQ            EQU C'S'       Sequential
+DFBAMODERANDOM         EQU C'R'       Random
+DFBAHOLDINDIC          DC C' '        Used for WPREAD operations
+                                      to specify that a subsequent
+                                      WPDELETE or WPREWRITE operation
+*                                     may be requested.
+DFBAHOLDUPDATE         EQU C'H'       Hold text for possible update
+                                      or deletion.
+DFBASEARCHMODE         DC C'S'        Used to specify to WPSEARCH
+                                      whether a specific or a Ggneric
+                                      search is to be performed.
+DFBASEARCHSPECF        EQU C'S'       Specific search
+DFBASEARCHGENRL        EQU C'G'       Generic (general) search
+DFBASPARE08            DC XL12'0'     [Reserved]
+DFBASPARE09            DC XL36'0'     [Reserved]
+DFBADOCUMENTINFO       DS 0XL384      Summary / header information
+                                      stored within the document's
+                                      "admin" block.
```

```
+*                                  Upon successful completion of
+*                                  WPOPEN, these fields contain
+*                                  either current information
+*                                  (existing document) or the
+*                                  established initial values for
+*                                  these fields if new document
+*                                  (For "D" default output mode,
+*                                  some initial values, including
+*                                  the print defaults, are obtained
+*                                  from the library's prototype
+*                                  document).
+DFBADOCIDENTITY        DS 0XL256   Internal descriptive
+                                   identification of document:
+DFBANAME               DC CL25' '  Document Name
+DFBASPARE10            DC CL15' '  [Reserved]
+DFBAOPERATOR           DC CL20' '  Operator
+DFBASPARE11            DC CL20' '  [Reserved]
+DFBAAUTHOR             DC CL20' '  Author
+DFBASPARE12            DC CL20' '  [Reserved]
+DFBACOMMENTS           DC CL20' '  Comments
+DFBASPARE13            DC CL20' '  [Reserved]
+DFBASPARE14            DC CL96' '  [Reserved]
+DFBADTWORKKEYS         DS 0XL30    Fields with creation / revision
+                                   date / time stamps and work
+                                   statistics.
+DFBACREATE             DS 0XL15    Statistics for new document:
+                                   (prior to first print request)
+DFBADATECR             DC PL4'0'   Creation date (0mmddyyF)
+DFBATIMECR             DC PL3'0'   Creation time (0hhmmF)
+DFBAWORKTIMECR         DC PL4'0'   Creation work time (0hhhhmmF)
+DFBAKEYSCR             DC PL4'0'   Creation keystrokes
+DFBAREVISED            DS 0XL15    Statistics for document
+                                   revision (after first print
+                                   request)
+DFBADATERE             DC PL4'0'   Revision date (0mmddyyF)
+DFBATIMERE             DC PL3'0'   Revision time (0hhmmF)
+DFBAWORKTIMERE         DC PL4'0'   Revision work time (0hhhhmmF)
+DFBAKEYSRE             DC PL4'0'   Revision keystrokes
+DFBAWORKTOTALS         DS 0XL18    Work and statistical totals for
+                                   document.
+DFBAWORKTIMETO         DC PL4'0'   Total work time (0hhhhmmF)
+DFBAKEYSTOTAL          DC PL4'0'   Total keystrokes
+DFBALINESTOTAL         DC PL4'0'   Estimated line count (need NOT
+                                   be filled in; set by WP Editor)
+DFBAPAGESTOTAL         DC PL3'0'   Count of total number of pages
+                                   in the document (not including
+                                   header, footer, and work pages)
+*                                  - updated by WPDELETE, WPREWRIT,
+*                                  and WPWRITE upon successful
+*                                  completion of requests.
```

```
+DFBALINESPAGE          DC PL3'0'       Lines / page count used for
+                                       repagination functions,
+                                       including the WP editor's
+*                                      (-COMMAND-)(-PAGE-) function.
+DFBALASTPRINTED        DS 0XL7         Date / time stamp of last,
+                                       if any, printing of document.
+DFBADATEPR             DC PL4'0'       Last printing date (0mmddyyF)
+DFBATIMEPR             DC PL3'0'       Last printing time (0hhmmF)
+DFBALASTARCHIVED       DS 0XL12        Date / time stamp and archive ID
+                                       from last, if any, archiving
+                                       of document.
+DFBADATEAR             DC PL4'0'       Last archiving date (0mmddyyF)
+DFBATIMEAR             DC PL3'0'       Last archiving time (0hhmmF)
+DFBAARCHIVEID          DC CL5' '       Last archive diskette's ID
+DFBAPASSWORD           DC CL6' '       Document password - WP Editor
+                                       protection mechanism.
+DFBAPASSWORDNULL       EQU C' '        Null password value
+DFBASPARE15            DC XL2'0'       [Reserved]
+DFBAGLOSSARYOPT        DC CL1' '       Glossary option indicator - if
+                                       set by WPOPEN, indicates that
+                                       document was previously verified
+*                                      as a glossary.
+DFBAGLOSSARY           EQU C'G'        Document is a verified glossary.
+DFBASPARE16            DC  XL51'0'     [Reserved]
+DFBAINFOUPDATE         DC  C'Y'        If the document is open for any
+                                       mode other than input and this
+                                       field is set to "Y", the
+*                                      document will be updated with
+*                                      the appropriate fields from
+*                                      DFBDOCUMENTINFO and
+*                                      DFBPRINTPARAMS when WPCLOSE is
+*                                      called.
+DFBAINFOUPDATEY        EQU C'Y'        Yes, update document information
+DFBAPRINTPARAMS        DS  0XL96       Information used both as
+                                       printing defaults for a
+                                       document and as
+*                                      WPPRINT entry to submit document
+*                                      print requests.  Except where
+*                                      noted, these fields are updated
+*                                      by WPCLOSE as per value of
+*                                      DFBINFOUPDATE.
+*                                      When calling WPPRINT, these
+*                                      parameters can be used
+*                                      individually as follows:
+*                                       - Valid values provided therein
+*                                      are used for the print request.
+*                                       - Numeric values of -1 and
+*                                      alphanumeric spaces or X'00'
+*                                      indicate that the parameters are
+*                                      to be taken from the document's
+*                                      own print defaults or from
+*                                      user task's defaults or from both.
```

```
+DFBAPRINTRANGES        DS  0XL15     Variables giving range
+                                     parameters (for pages) for
+                                     print requests.
+DFBAFROMPAGE           DC  PL3'0'    First page to be printed.
+DFBATHRUPAGE           DC  PL3'0'    Last page to be printed.
+DFBASTARTPAGE          DC  PL3'0'    For page numbering, first
+                                     to be used in headers / footers
+DFBA1STHEADERP         DC  PL3'0'    First page for headers, if any.
+DFBA1STFOOTERP         DC  PL3'0'    First page for footers, if any.
+DFBAPRINTLINES         DS  0XL6      Variables specifying line
+                                     count specifications.
+DFBA1STFOOTERL         DC  PL3'0'    First line on page for footers,
+                                     if any.
+DFBAPAPERLENGTH        DC  PL3'0'    Number of lines per physical
+                                     form.
+DFBAPRINTMARGINS       DS  0XL4      Variables specifying left
+                                     margin print specifications.
+DFBAMARGIN1            DC  PL2'0'    Number of characters in left margin
+DFBAMARGIN2            DC  PL2'0'    Number of characters in left margin
+                                     for secondary document for
+                                     dual-column print requests
+DFBACOPYCOUNT          DC  PL3'0'    Copy count.
+DFBAFORMNUMBER         DC  PL2'0'    VS printing form number or WP
+                                     Printer device number, 0 - 254
+DFBACHARACSET1         DC  PL2'0'    Character set number, 0 - 9
+DFBACHARACSET2         DC  PL2'0'    Character set number, 0 - 9,
+                                     for dual-wheel printers and for
+                                     secondary document in
+*                                    dual-column print requests.
+DFBASPARE17            DC  XL14'0'   [Reserved]
+DFBAPRINTSCHED         DS  0XL16     Print request scheduling
+                                     parameters.
+DFBAPRINTAREA          DC  CL8' '    Print area (not stored as
+                                     default in document).
+DFBAPRINTCLASS         DC  CL1' '    Print class.
+DFBASPARE18            DC  XL5'0'    [Reserved]
+DFBAPRINTTYPE          DC  CL1'N'    Type of print request (not
+                                     stored as default in document).
+DFBAPRTNORMAL          EQU C'N'      Normal print request.
+                                     (single document, single column)
+DFBAPRTMERGE           EQU C'M'      Merge print request,
+DFBAPRTDUAL            EQU C'D'      Dual column print request
+                                     (one or two documents).
+DFBADISPOSITION        DC  C'S'      After-print document disposition
+DFBAPRTSAVE            EQU C'S'      Save document.
+DFBAPRTDELETE          EQU C'D'      Delete document.
+DFBAPRINTSTYLE         DS  0XL16     Print request style and format
+                                     specification parameters.
+DFBAFORMSTYPE          DC  C'S'      Forms type.
+DFBASTANDARD           EQU C'S'      Standard.
```

```
+DFBACONTINUOUS         EQU C'C'       Continuous forms.
+DFBAFORMS1             EQU C'1'       Forms / bin 1.
+DFBAFORMS2             EQU C'2'       Forms / bin 2.
+DFBAHORIZONTAL         DC  C'10'      Horizontal pitch.
+DFBA10PITCH            EQU C'10'      10 pitch.
+DFBA12PITCH            EQU C'12'      12 pitch.
+DFBA15PITCH            EQU C'15'      15 pitch.
+DFBAPSPITCH            EQU C'PS'      Proportional spacing pitch.
+DFBAVERTICAL           DC  C'06'      Vertical pitch (not stored as
+                                      default in document).
+DFBA06VPITCH           EQU C'06'      6 lines per inch.
+DFBA08VPITCH           EQU C'08'      8 lines per inch.
+DFBAFINALDRAFT         DC  C'F'       Final/draft specification.
+DFBAFINAL              EQU C'F'       Final.
+DFBADRAFT              EQU C'D'       Draft (doubled spaced).
+DFBAFORMAT             DC  C'JU'      Justification specification.
+DFBAUNJUSTIFIED        EQU C'UN'      Unjustified.
+DFBAJUSTIFIED          EQU C'JU'      Justified.
+DFBAWITHNOTES          EQU C'NO'      With notes, unjustified.
+DFBADOCSUMMARY         DC  C'Y'       Document summary print option
+                                      specification.
+DFBASUMMARYYES         EQU C'Y'       Yes, print document summary.
+DFBASUMMARYNO          EQU C'N'       No, do not print summary.
+DFBASPARE19            DC  XL7'00'    [Reserved]
+DFBAFULLDOCID2         DS  0CL24      These fields are utilized to
+                                      identify the document to be
+                                      used as a secondary document
+*                                     for a merge or dual-column
+*                                     print-request.
+DFBADOCUMENTID2        DC  CL4' '     Must have a 4-digit number as
+                                      value unless no secondary
+                                      document, in which case spaces
+*                                     should be utilized.
+DFBASPARE20            DC  CL4' '     [Reserved]
+DFBALIBRARY2           DC  CL1' '     Document library must always
+                                      be provided.
+DFBASPARE21            DC  CL7' '     [Reserved]
+DFBAVOLUME2            DC  CL6' '     Any program-supplied value in
+                                      this field will override the
+                                      document library's default
+*                                     volume value.
+DFBASPARE22            DC  CL2' '     [Reserved]
+DFBASPARE24            DC  XL152'0'   [Reserved]
+DFBALIBRARYLIST        DS  0XL256     Fields used by the WPDOCLIB
+                                      entry for obtaining the start
+                                      document and returning the total
+*                                     document count and the document
+*                                     list to the calling program.
+DFBALIBRTOTAL          DC  H'0'       Total number of documents
+                                      found in library.
```

```
+DFBALIBRSTART          DC  H'1'       Number of the first document to
+                                      be listed.
+DFBASPARE25            DC  XL12'0'    [Reserved]
+DFBALIBRENTRIES        DS  0XL240     Area defined for aggregate of
+                                      all 30 entries.
+DFBALIBRENTRY          DS  30XL8      Individual entry aggregate.
+                       ORG DFBALIBRENTRY
+DFBALIBRDOCUMENT       DS  CL4        Document in library.
+DFBASPARE26            DC  XL4'0'     [Reserved]
+                       ORG
+                       ORG
+*       The following provides equates for the values of the
+*       formatting characters found in the text of the document
+DFBACENTER             EQU X'01'
+DFBATAB                EQU X'02'
+DFBARETURN             EQU X'03'
+DFBAINDENT             EQU X'04'
+DFBADECIMALTAB         EQU X'05'
+DFBAFORMATLINE         EQU X'06'
+DFBASTOP               EQU X'0B'
+DFBANOTE               EQU X'0C'
+DFBAMERGE              EQU X'0D'
+DFBASUPERSCRIPT        EQU X'0E'
+DFBASUBSCRIPT          EQU X'0F'
+DFBANEWPAGE            EQU X'86'
+DFBASTOPX              EQU X'8B'
+DFBANOTEX              EQU X'8C'
+DFBAMERGEX             EQU X'8D'
+DFBASUPERSCRIPTX       EQU X'8E'
+DFBASUBSCRIPTX         EQU X'8F'
+DFBAEND                EQU *
+DFBALENGTH             EQU DFBAEND-DFBA
+BEGIN                  CODE
```

4.2.20 DISMOUNT - Dismount Disk or Tape Volume (SVC 41)

Syntax

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

                 NODISPLAY={YES} [,NOWAIT={YES}][,VSID={(register)}
                           {NO }          { NO}       { 'string' }
                                                      { address  }
```

Function

Requests the logical dismounting of a disk or tape volume. If the volume referenced is a tape volume, it is also rewound and unloaded. If the disk is the root disk of a multidisk volume set and its VTOC shows any open files, the disk can not be dismounted.

Parameter Definitions

VOLUME — The name of the volume which is to be dismounted. 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 parameter is required.

TYPE — Indicates whether the volume is a disk or a tape volume. Valid values are DISK and TAPE. This parameter is optional. The default is DISK.

NODISPLAY — YES 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.

NOWAIT — YES indicates that the calling program will not wait for an answer back from the system task after issuing a dismount. No messages are displayed on the workstation (implied NODISPLAY) or on the operator screen. The default is NO.

VSID — The volume set identification number for a volume which is part of a volume set. If no outstanding I/O exists on a non-root set member or on a root volumes, if no files are open on a single volume, the volume is dismounted.

Stack On Input

```
              |                          |    Lower
              |                          |    Address
              |0____1______2______3______|
     0(SP)    |      |      |            |
              | (1)  |(2)   |(3)         |
              | FLAG |      |VOL NAME    |
     4(SP)    |__________________________|
              |                          |
              |                          |    Higher
              |__________________________|    Address
              |        Preceding         |
              |        Stack Data        |
```

(1) Flag

```
        Bit 0    0 = Dismount disk
                 1 = Dismount tape
        Bit 1    1 = No display option.
                     Do not write to caller's workstation
        Bit 2    1 = No wait option
                     Do not write dismount message, tell SYSTASK to
                     erase dismount message.
```

(2) Volume set ID

(3) Volume name (6 bytes)

Stack On Output

```
                                              Lower
              |                          |    Address
              |__________________________|
     0(SP)    |                          |
              |  (1)  Return Code        |    Higher
              |__________________________|    Address
              |        Preceding         |
              |        Stack Data        |
```

(1) Return code

Output

DISMOUNT issues a return code to the user program in the topword of the stack as follows:

Code	Description
0	Success.
4	Input volume name is blank, or bytes 0-1 in input are nonzero.
8	Volume not found.
12	Volume can not be dismounted.
16	Device detached.
20	Volume in use by a user or the operating system.
24	Volume reserved by another user.
28	GETMEM pool failure.
32	Device is reserved by another task.
36	IPC error.
40	Volume cannot be mounted, I/O in progress.
44	Success, but VSCB deallocation failed.

Examples

```
 DMO      DISMOUNT VOLUME='VOL444',TYPE=DISK
+DMO      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

 DM1      DISMOUNT VOLUME=(R4)
+DM1      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
```

```
 DM2        DISMOUNT VOLUME=TAPEVOL,TYPE=TAPE
+DM2        PUSHN 0,8              GET TWO WORDS 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


   LAB      DISMOUNT   VOLUME=TAPEVOL,TYPE=TAPE
+  LAB      PUSHN 0,8                       GET TWO WORDS 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
```

4.2.21 ENDLOCAL - End Generation of Local Symbol Names

Syntax

    ENDLOCAL

Function

Turns off the automatic generation of local symbol names initiated by the use of the LOCAL macro.

Example

See the use of this macro in the example under LOCAL.

4.2.22 EXTRACT - Extract Data From System Control Blocks (SVC 28)

Syntax

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

        [,ALOGFTIME=address][,ASUPPORT=address] [,ATOETRT=address]
        [,CDISKET=address]  [,CPU=address]      [,CURLIB=address]
        [,CURVOL=address]   [,CURRENCY=address] [,DATEFMT=address]
        [,DATESEP=address]  [,DECIMALPT=address][,DEVCNT=address]
        [,DISKIO=address]   [,DYVAL=address]    [,ETOATRT=address]
        [,EXFLGS=address]   [,EXTPRIOR=address] [,ETIME=address]
        [,FILECLAS=address] [,FORM#=address]    [,HZ=address]
        [,INLIB=address]    [,INVOL=address]    [,JOBCLASS=address]
        [,JOBLIMIT=address] [,JOBNAME=address]  [,JOBQUEUE=address]
        [,LINES=address]    [,NATION=address]   [,NRES=address]
        [,OCNT=address]     [,OTIO=address]     [,OUTLIB=address]
        [,OUTVOL=address]   [,PCPCW=address]    [,PICOUNT=address]
        [,POCOUNT=address]  [,PRINTER=address]  [,PRINTIO=address]
        [,PRNTMODE=address] [,PROGLIB=address]  [,PROGVOL=address]
        [,PRTCLASS=address] [,PTIME=address]    [,RDFLGS=address]
        [,RUNLIB=address]   [,RUNVOL=address]   [,SEG2BUF=address]
        [,SEG2SIZE=address] [,SEG2SZE=address]  [,SICOUNT=address]
        [,SOCOUNT=address]  [,SPOOLIB=address]  [,SPOOLSYS=address]
        [,SPOOLVOL=address] [,STACK=address]    [,STATIC=address]
        [,SYSID=address]    [,SYSLIB=address]   [,SYSNAME=address]
        [,SYSPAGE=address]  [,SYSVOL=address]   [,SYSWORK=address]
        [,TAPEIO=address]   [,TASK#=address]    [,TASKTYPE=address]
        [,THOUSSEP=address] [,TIMESEP=address]  [,UEXFLGS=address]
        [,URDFLGS=address]  [,USERID=address]   [,USERNAME=address]
        [,UWTFLGS=address]  [,VERSION=address]  [,VOICEIO=address]
        [,WORKLIB=address]  [,WORKVOL=address]  [,WS=address]
        [,WSIO=address]     [,WTFLGS=address]   [,PRTFILECLAS=address]

        [,ALOGFENAB=(inaddress,outaddress)]
        [,CLUSTER=(inaddress,outaddress)]
        [,DEVICE=(inaddress,outaddress)]
        [,DLPNAME=(inaddress,outaddress)]
        [,TAPEVOL=(inaddress,outaddress)]
        [,VOLVCB=(inaddress,outaddress)]

        [,DEVLIST=(inaddress,outaddress,outarealength)]
        [,DLPDEV#=(inaddress,outaddress,outarealength)]
        [,OTASK=(inaddress,outaddress,outarealength)]
        [,VOLUME=(inaddress,outaddress,outarealength)]
```

Function

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

Parameter Definitions

FORM Describes the type of information required.

BRIEF Used to request four items as described below. The output area must be at least 12 bytes long.

(1) Amount of memory, in bytes, that is currently not fixed (4 bytes).
(2) Number of files that a task may have open simultaneously (2 bytes).
(3) Workstation number associated with requesting task, or -1 if no associated workstation (2 bytes).
(4) Remaining stack space, in bytes, after return from EXTRACT (4 bytes).

FULL Used to request all the items listed below. The output area must be at least 98 bytes long.

(1) Total physical area in bytes not currently resident (4 bytes).
(2) Number of files that a task may have open simultaneously (2 bytes).
(3) Workstation number associated with requesting task, or -1 if no associated workstation (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 volume name (6 bytes).
(7) System default library name (8 bytes).
(8) Task's default printer number, or -1 if no default printer number (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 (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 the user modifiable area 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 in the format VVRRPP, where VV is the version, RR is the revision, and PP is the patch level) (3 bytes).

PCPCW — Program control word (PCW) at the time of the most recent program exception for which a user exit was specified (8 bytes). PCPCW is used to request the value of the current PCW when a program exception occurs for which an exit routine was provided, and 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.

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.

LIST — Used when a list of needed items is supplied. The parameter specifies the address of an area to receive the corresponding data item.

To use any of the parameters described below, FORM=LIST must be specified.

ALOGFTIME — Automatic logoff time interval is used to obtain the maximum time that a workstation may remain inactive before it is automatically logged off. Time may be from 0 to 99 minutes. Returns one halfword with the time in the least significant byte (2 bytes).

ASUPPORT — Application support is used to determine if WP, Mailway®, and DMS/TX are supported on the system (1 byte).

ATOETRT — ASCII-to-EBCDIC translation table (256 bytes). See TR instruction in <u>VS Principles of Operation</u> manual for use.

CDISKET — Device number of system's central diskette (2 bytes).

CPU Current CPU ID (2 bytes).

CURLIB Library in which current program resides (8 bytes).

CURRENCY Currency symbol (3 bytes).

CURVOL Volume where current program resides (6 bytes).

DATESEP Date separator (1 byte).

DATEFMT Indicates whether the date set is in European or American format (1 byte).

DECIMALPT Decimal point character (1 byte).

DEVCNT Highest device number in device configuration (4 bytes).

DISKIO Count of disk I/Os for this run (4 bytes).

DYVAL One day in clock units (4 bytes).

ETIME Elapsed run time since command processor initiation, in hundredths of seconds (4 bytes).

ETOATRT EBCDIC-to-ASCII translation table (256 bytes). See TR instruction in VS Principles of Operation manual for use.

EXFLGS Current file-access bit map for execute access, from program file block (PFB)(4 bytes).

EXTPRIOR Task's current paging priority from task control block (1 byte).

FILECLAS Default output file-access protection class, or blank (1 byte).

FORM# Default form number for print files (0-254) (1 byte).

HZ A/C line frequency (2 bytes).

INLIB Default input library (8 bytes).

INVOL Default input volume for OPEN (6 bytes).

JOBCLASS Default job class (A-Z) (1 byte).

JOBLIMIT Default job CPU time limit (4 bytes).

JOBNAME Name of background job (8 bytes).

JOBQUEUE Default job status: Run (R) or Hold (H) (1 byte).

LINES Suggested lines-per-page for print files (1 byte).

NRES Total physical area not currently resident, in bytes (4 bytes).

NATION Nation code (1 byte)

OCNT Number of files that current task can have open simultaneously, excluding files already open (2 bytes).

OTIO Count of I/Os for other devices not included under WSIO, DISKIO, PRINTIO, or TAPEIO (4 bytes).

OUTLIB Default output library (8 bytes).

OUTVOL Default output volume for OPEN (6 bytes).

PCPCW Program Check Old PCW for last program check (8 bytes).

PICOUNT Program pagein count (4 bytes).

POCOUNT Program pageout count (4 bytes).

PRINTER Task's default printer number, or -1 if no default printer (2 bytes).

PRINTIO Count of printer I/Os for this run (4 bytes).

PRNTMODE Default print output mode (1 byte).

PROGLIB User program library used by LINK SVC (8 bytes).

PROGVOL User program volume name used by LINK SVC (6 bytes).

PRTCLASS Default print class for print files (A-Z) (1 byte).

PRTFILECLAS Default file class for print class (1 byte).

PTIME Processor time of run since command processor initiation, in hundredths of seconds (4 bytes).

RDFLGS Current file-access bit map for read access, from program file block (PFB) (4 bytes).

RUNLIB User program library name, used by command processor RUN function (8 bytes).

RUNVOL User program library volume, used by command processor RUN function (6 bytes).

SEG2BUF Number of the user modifiable area buffer pages currently available (2 bytes).

SEG2SIZE Length of the user modifiable area, in bytes (4 bytes).

SEG2SZE Default the user modifiable area size for any task in the system (2 bytes).

SICOUNT System pagein count (4 bytes).

SOCOUNT System pageout count (4 bytes).

SPOOLIB Spool library name constructed from user ID or background task number (8 bytes).

SPOOLSYS Remote system to which print files are automatically routed via file transfer service (8 bytes).

SPOOLVOL Default spool volume (6 bytes).

STACK Remaining stack space in bytes after return from EXTRACT (4 bytes).

STATIC Pointer to beginning of static areas for current program (4 bytes). This pointer may be useful in re-establishing the ability to address in a CEXIT routine.)

SYSID System WangNet name (8 bytes).

SYSLIB System default library name (8 bytes).

SYSNAME System name (16 bytes).

SYSPAGE System paging library name (8 bytes).

SYSVOL System default library volume name (6 bytes).

SYSWORK System work library (paging files, system task queues, etc.) which BACKUP skips (8 bytes).

TAPEIO Count of tape I/Os this run (4 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).

THOUSSEP Thousands separator (1 byte).

TIMESEP Time separator (1 byte).

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).

USERID User logon identification (3 bytes).

USERNAME User name (from system user list) (24 bytes).

UWTFLGS User's base file-access bit map for update access, from user's extended task control block (ETCB) (4 bytes).

VERSION Operating system version number, which is a packed number in the format VVRRPP, where VV is the version, RR is the revision, and PP is the patch level (3 bytes).

VOICEIO The number of voice device I/Os for this run (1 byte).

WORKLIB Work library name constructed from user ID or background task number (8 bytes).

WORKVOL Default work volume (6 bytes).

WS Workstation number associated with requesting task, or -1 if no associated workstation (2 bytes).

WSIO Count of workstation I/Os for this run (4 bytes).

WTFLGS Current file-access bit map for update access, from program file block (PFB) (4 bytes).

For the following seven parameters, two addresses are supplied. The first address specifies further input, and the second address specifies an area to receive the corresponding data. To use these parameters, FORM=LIST must be specified.

ALOGFENAB Input: Device number in hex (1 byte).
Output: Automatic logoff enable status (1 byte).
Y = YES, N = NO, hex 0 = invalid device number.

CLUSTER

Input: Device number (2 bytes).

Output: (1) Device number of the archiver diskette, or 0 if none (2 bytes).
(2) Device number of the next device on the cluster, or 0 if none (2 bytes).
(3) IOP port (1 byte). The value is taken from the UCBDRTEPORT field of the unit control block (UCB).
(4) Broadband channel (1 byte). The value is taken from the UCBDRTECHL field of the UCB.
(5) Short address (1 byte). The value is taken from the UCBDRTEASA field of the UCB.
(6) Device on the cluster (1 byte). The value is taken from the UCBDRTEDCC field of the UCB.
(7) Cluster on port (1 byte). The value is taken from the UCBDRTECC field of the UCB.
(8) Spare (7 bytes).

NOTE

The CLUSTER parameter is used to obtain the device number of the archiver diskette on the same cluster as the input device number. If more than one archiver diskette is on the cluster, then the archiver diskette device number returned is the next in sequence.

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 no task identifier (4 bytes).
(5) Volume name of removable volume, disk or tape only (6 bytes). Blank if nothing mounted.
(6) Volume name of fixed volume, disk only (6 bytes). Blank if nothing mounted.
(7) Density support for tapes (2 bytes). First byte = T for triple density, = D for double density. Second byte = G for 6250 BPI (GCR mode). For disks, this value represents the following: VSID for removable (1 byte); VSID for fixed (1 byte).
(8) Physical device address (2 bytes)

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

NOTE

The output area contains 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) controlled by the DLP (1 byte).
(5) Microcode file status: X'00' if stopped, X'80' if loaded (1 byte).
(6) DLPKIND '0' = peripheral/device processor, '1' = device processor (1 byte).
(7) Reserved for future use (2 bytes).
(8) Microcode file name (8 bytes), zero if not loaded.
(9) Microcode library name (8 bytes), zero if not loaded.
(10) Microcode volume name (6 bytes), zero if not loaded.
(11) Reservation status of DLP (1 byte): X'80' if reserved, X'00' if not reserved.
(12) Task number of the task which reserved the DLP (3 bytes).
(13) Current protocol ID (2 bytes).
(14) Loadable status: X'00' nonloadable
X'01' loadable

IOPTYPE Input: Device address (1 byte).
Output: (1) IOP/IOC type (1 byte). The value is taken from the PPBTYPE field of the peripheral processor block (PPB) for the IOP associated with the input device address.
(2) Spare (11 bytes).

TAPEVOL Input: Volume serial number (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 no task ID (4 bytes).
(7) Current file sequence number (2 bytes).
(8) 6 bytes of binary zeroes (reserved).

VOLVCB Input: Volume name (6 bytes).
Output: (1) VCB address (4 bytes).
(2) Reserved (4 bytes).

The next four parameters each have three subparameters. The first subparameter specifies the address of further input, the second subparameter specifies the address of an area to receive the corresponding data, and the third subparameter is the length of the output area (specified as an expression or register in parentheses). The maximum number of device addresses in the device list is 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).

DLPDEV# Input: Device address (2 bytes).
Output: (1) Device status flag (1 byte): 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 configured (4 bytes).

NOTE

For the DLPDEV# parameter, the output area contains zeroes if the specified device address is invalid.

OTASK

Input: Task identifier (4 bytes).

Output: (1) Workstation device number of task specified, or -1 if no associated workstation (1 byte).
(2) Current user ID for task specified, or blank if no associated user ID (3 bytes).
(3) Current user name for task specified, or blank if no associated user name (24 bytes).
(4) Type (F, FS, B, BS) of task specified (see TASKTYPE) (2 bytes).
(5) Status of task (18 bytes).
(6) Name of initial program run (8 bytes).
(7) Name of current program run (8 bytes).
(8) Initial program start date (4 bytes).
(9) Initial program start time (4 bytes).
(10) Elapsed time since CP initiation, in hundredths of seconds (4 bytes).
(11) Processor time since CP initiation, in hundredths of seconds.
(12) Count of workstation I/O since initial program run (4 bytes).
(13) Count of disk I/O since initial program run (4 bytes).
(14) Count of tape I/O since initial program run (4 bytes).
(15) Count of printer I/O since initial program run (4 bytes).
(16) Count of other I/O since initial program run (4 bytes).
(17) Program pagein count (4 bytes).
(18) Program pageout count (4 bytes).
(19) System pagein count (4 bytes).
(20) System pageout count (4 bytes).
(21) Reserved (4 bytes).

NOTE

The VOLUME parameter cannot be used for volume sets. Use VSETINFO or VOLINFO.

VOLUME

Input: Volume name (6 bytes).

Output: (1) Device number (1 byte), or -1 if volume not mounted.

(2) Volume type (1 byte): F for fixed, R for removable, or blank if not mounted.

(3) Label type (2 bytes): SL (standard label), NL (no label), or blank if not mounted.

(4) Usage (2 bytes): SH (shared), RR (restricted removal), PR (protected), EX (exclusive), or blank.

(5) Task identifier always -1 (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 replacement or unused blocks (2 bytes).

(10) Number of files open on this volume (2 bytes).

(11) Sector type, diskette only (1 byte): soft sector (S), hard sector (H).

(12) Addressing in effect, diskette only (1 byte): nonstandard (N), standard (S).

(13) Fault tolerance level (2 bytes): (CT) crash tolerant, (MT) media tolerant, or blank for no tolerance.

(14) Paging file eligibility (1 byte): (Y) paging files allowed, (N) no paging files allowed.

(15) Spool file eligibility (1 byte): (Y) spool files allowed, (N) no spool files allowed.

(16) Work file eligibility (1 Byte): (Y) work files allowed, (N) no work files allowed.

(17) Secure volume (1 byte): (Y) volume is secure, (N) volume is not secure.

(18) Cylinders per diagnostic volume (4 bytes). Includes all of EXTRDVOLCVP plus diagnostic cylinders.

(19) Extent limit for file creations (1 byte).

(20) Total extent limit (1 byte).

Examples

```
 EX0        EXTRACT    FORM=BRIEF,AREA=(R3)
+EX0        PUSH       0,R3          AREA
+           MVI        0(15),0       FORM=BRIEF
+           SVC        28 (EXTRACT)
```

```
 EX1        EXTRACT   INLIB=A1,INVOL=(R1)
+EX1        DS        0H
+           PUSH      0,R1         INVOL
+           PUSHA     0,11         IDENTIFIER
+           PUSHA     0,A1         INLIB
+           PUSHA     0,12         IDENTIFIER
+           PUSHA     0,2          COUNT OF ITEMS
+           MVI       0(15),3      FORM=LIST
+           SVC       28 (EXTRACT)


 EX2        EXTRACT   OUTLIB=A1,VOLUME=(A2,(R1))
+EX2        DS        0H
+           PUSHA     0,A1         OUTLIB
+           PUSHA     0,15         IDENTIFIER
+           PUSHA     0,1          COUNT OF ITEMS
+           MVI       0(15),3      FORM=LIST
+           SVC       28 (EXTRACT)

  EX3       EXTRACT DEVLIST=(A2,(R1),12)
+EX3        DS    0H
+           PUSHA 0,A2             DEVLIST INPUT
+           PUSH  0,R1             DEVLIST OUTPUT
+'          MVI   0(15),12         OUTPUT LENGTH
+           PUSHA 0,59             IDENTIFIER
+           PUSHA 0,1              COUNT OF ITEMS
+           MVI   (15),4           FORM=LIST WITH ADDITIONAL INPUT
+           SVC   28 (EXTRACT)
```

4.2.23 EXTRD - Describe Output Area For The Extract SVC

Syntax

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

Function

Describes the data structure used by the EXTRACT supervisor call to store the values of the requested information.

Parameter Definitions

NODSECT — Specification of NODSECT results in the EXTRD fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, the system generates a DSECT with the name EXTRD (plus the optional suffix).

REG — Provides the optional specification of a register for which a USING statement for the EXTRD fields is generated.

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

Structure

```
                        BYTE 0    BYTE 1    BYTE 2    BYTE 3

        EXTRD
BEGIN
CLASS0  |  +0   | NRES                                    |
           +4   | OCNT              | WS                  |
           +8   | STACK                                   |

ORG

CLASS1  |  +0   |                                         |
           +4   |                                         |
           +8   |                                         |
           +C   | DYVAL                                   |
SYSVOL  | +10   | SCDVOL                                  |
          +14   |                   | SCDNAME             |  +16 ↕ SYSLIB
          +18   |                                         |
          +1C   |                   | DEFPRT              |  +1E ↕ PRINTER
RUNVOL  | +20   | UPDVOL                                  |
          +24   |                   | UPDNAME             |  +26 ↕ RUNLIB
          +28   |                                         |
          +2C   |                   | EXLFGS              |
          +30   |                   | VOL                 |  +32 ↕ INVOL
          +34   |                                         |
```

		BYTE 0	BYTE 1	BYTE 2	BYTE 3	
INLIB	+38	FILE1				
	+3C					
	+40	RDFLGS				
OUTVOL	+44	VOLO				
	+48			FILE1O		+4A ↕ OUTLIB
	+4C					
	+50			WTFLGS		
	+54			SEG2BUF		
PRNTMODE	+58	PRTTYPE	FPCLASS	USERID	TCBSCC	+59 ↕ FILECLAS
	+5C		EXTPRIOR	LINES	VERSION	+5D ↕ TCBSCC
	+60			DEFLIB1N		+5F ↕ SPARE1
	+64					
	+68			DEFLIB1V		
				.		
				.		
				.		
	+33A					

ORG

CLASS2	+0	PCPCW			
	+4				

ORG

+0	DEVCLASS	TYPE	DEVUSAGE	
+4	DEVUSER			
+8	DEVREM			
+C			DEVFIXED	
+10				
+14	DEVSPEC	DEVSPEC2	DEVPDA	

ORG

+0	VOLDEV	VOLTYPE	VOLLABEL	
+4	VOLUSAGE		VOLUSER	
+8			VOLBC	
+C	VOLMAXTFR		VOLCV	
+10	VOLCVP		VOLOCNT	
+14	VOLSECT	VOLADDR	VOLTOL	
+18	VOLPAGE	VOLSPOOL	VOLWORK	VOLSECURE
+1C	VOLCVD		XLMTOPEN	XLMTTOTL

ORG

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+0	OTASKWS	OTASKUID		
+4	OTASKNAME			
+8				
+C				
+10				
+14				
+18				
+1C	OTASKTYPE		OTASKSTAT	
+20				
+24				
+28				
+2C				
+30	OTASKPROGI			
+34				
+38	OTASKPROGC			
+3C				
+40	OTASKIDATE			
+44	OTASKITIME			
+48	OTASKETIME			
+4C	OTASKPTIME			
+50	OTASKWSIC			
+54	OTASKDSKIO			
+58	OTASKTAPIO			
+5C	OTASKPRTIO			
+60	OTASKVCIO			
+64	OTASKOTIO			
+68	OTASKPICNT			
+6C	OTASKPOCNT			
+70	OTASKSICNT			
+74	OTASKSOCNT			
+78	OTASKSPARE			

ORG

+0	TAPEDEV	TPESPAR1	TAPEDEN	
+4	TAPELABEL		TAPEUSAGE	
+8	TAPEUSER			
+C	TAPEFSEQ		TAPESPAR2	
+10				

ORG

+0	DLTOT	DLNUM	DLENTRY		+2 ↕ DLIST

ORG

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+0	DLPDEVMAP			
+4	DLPDEV#1		DLPTYPE	DLPLINECNT
+8	DMCSTATUS	DLPKIND	DLPSPARE	
+C	MCFILE			
+10				
+14	MCLIB			
+18				
+1C	MCVOL			
+20			DLPRSRV	DLPTASK#
+24				

ORG

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+0	DEVSTATUS	DEVTASK#		
+4	DEVDLPNAME			

ORG

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+0	VCBADDR			
+4				

ORG

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+0	DEFLIBR			
.				
.				
.				
+44				

ORG

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+0	ADISKET		NEXTDEV	
+4	PORT	DCHNL	ASA	DCC
+8	COP	SPARE		
+C				
.				
.				
.				
+78	IDIDERR			

ORG

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+ 0	ALOGF			

Example

```
         EXTRD
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)
*
EXTRDBEGIN                   DS  0XL1        (UNALIGNED)
*
EXTRDIDMAX                   EQU   100       MAX ID # CURRENTLY IN
*                                            USE-FROM EXTRACT MACRO
***********************CLASS 0*****************************************
EXTRDCLASS0                  DS  0XL12       RETURNED FOR CLASS 0:
EXTRDNRES                    DS  AL4         PHYSICAL MEMORY (BYTES)
*                                              NOT PERMANENTLY RESIDENT
EXTRDOCNT                    DS  HL2         NUMBER OF FILES WHICH
*                                               CURRENT TASK MAY HAVE
*                                               OPEN, EXCLUDING FILES
*                                               ALREADY OPEN
EXTRDWS                      DS  HL2         TASK'S ASSOCIATED
*                                               WORKSTATION NUMBER, OR

*                                               -1 IF NONE
EXTRDSTACK                   DS  AL4         REMAINING STACK SPACE
*
***********************CLASS 1*****************************************
           ORG EXTRDBEGIN
EXTRDCLASS1                  DS  0XL98       RETURNED IN ADDITION
*                                               FOR CLASS 1:
           ORG EXTRDBEGIN+L'EXTRDCLASS0
EXTRDDYVAL                   DS  FL4         ONE DAY IN CLOCK UNITS
EXTRDSYSVOL                  DS  0CL6        SYSTEM DEFAULT LIBRARY
EXTRDSCDVOL                  DS  CL6            VOLUME NAME
EXTRDSYSLIB                  DS  0CL8        SYSTEM DEFAULT LIBRARY
EXTRDSCDNAME                 DS  CL8            NAME
EXTRDPRINTER                 DS  0HL2        DEFAULT ONLINE PRINTER
EXTRDDEFPRT                  DS  HL2            DEVICE NUMBER,
*                                               OR -1 OF NONE
EXTRDRUNVOL                  DS  0CL6
EXTRDUPDVOL                  DS  CL6         USER PROGRAM LIB.VOLUME
EXTRDRUNLIB                  DS  0CL8
EXTRDUPDNAME                 DS  CL8         USER PROGRAM LIB.NAME
EXTRDEXFLGS                  DS  BL4         'EXECUTE' ACCESS MASK
EXTRDINVOL                   DS  0CL6
```

```
EXTRDVOL                      DS  CL6         DEFAULT INPUT VOLUME
EXTRDINLIB                    DS  0CL8
EXTRDFILE1                    DS  CL8         DEFAULT INPUT LIBRARY
EXTRDRDFLGS                   DS  BL4         'READ' ACCESS MASK
EXTRDOUTVOL                   DS  0CL6
EXTRDVOLO                     DS  CL6         DEFAULT OUTPUT VOLUME
EXTRDOUTLIB                   DS  0CL8
EXTRDFILE1O                   DS  CL8         DEFAULT OUTPUT LIBRARY
EXTRDWTFLGS                   DS  BL4         'WRITE' ACCESS MASK
EXTRDSEG2BUF                  DS  BL2         NUMBER OF USER MODIFIABLE
*                                                AREA 'BUFFER' PAGES
*                                                CURRENTLY AVAILABLE
EXTRDPRNTMODE                 DS  0CL1
EXTRDPRTTYPE                  DS  CL1         PRINT OUTPUT MODE
*                                                ('S', 'H', OR 'O')
EXTRDFILECLAS                 DS  0CL1
EXTRDFPCLASS                  DS  CL1         DEFAULT FILE PROTECT
*                                                CLASS
EXTRDUSERID                   DS  CL3         CURRENT USER LOGON ID
EXTRDTCBSCC                   DS  0HL1        (DO NOT USE)
EXTRDEXTPRIOR                 DS  HL1         TASK'S PAGING PRIORITY
EXTRDLINES                    DS  HL1         SUGGESTED LINES/PAGE
EXTRDSPARE1                   DS  0BL3        UNUSED PRIOR TO RELEASE 3.1
*                                             (WAS BINARY ZEROES)
EXTRDVERSION                  DS  XL3         SYSTEM VERSION NUMBER
*                                             (SEE EXTRDIDVERSION)
EXTRDIDDEFLIB1                EQU 93,80,C
EXTRDIDDEFLIB2                EQU 94,80,C

EXTRDIDDEFLIB3                EQU 95,80,C
EXTRDIDDEFLIB4                EQU 96,80,C
EXTRDIDDEFLIB5                EQU 97,80,C
EXTRDIDDEFLIB6                EQU 98,80,C
EXTRDIDDEFLIB7                EQU 99,80,C
EXTRDIDDEFLIB8                EQU 100,80,C
EXTRDIDDEFLIB9                EQU 101,80,C
EXTRDIDDEFLIB10               EQU 102,80,C
EXTRDDEFLIB1N                 DS  CL8         DEFAULT LIB NAME #1
EXTRDDEFLIB1V                 DS  CL72        DEFAULT NAMESTRING 1
EXTRDDEFLIB2N                 DS  CL8         DEFAULT LIB NAME #2
EXTRDDEFLIB2V                 DS  CL72        DEFAULT NAMESTRING 2
EXTRDDEFLIB3N                 DS  CL8         DEFAULT LIB NAME #3
EXTRDDEFLIB3V                 DS  CL72        DEFAULT NAMESTRING 3
EXTRDDEFLIB4N                 DS  CL8         DEFAULT LIB NAME #4
EXTRDDEFLIB4V                 DS  CL72        DEFAULT NAMESTRING 4
EXTRDDEFLIB5N                 DS  CL8         DEFAULT LIB NAME #5
EXTRDDEFLIB5V                 DS  CL72        DEFAULT NAMESTRING 5
EXTRDDEFLIB6N                 DS  CL8         DEFAULT LIB NAME #6
EXTRDDEFLIB6V                 DS  CL72        DEFAULT NAMESTRING 6
EXTRDDEFLIB7N                 DS  CL8         DEFAULT LIB NAME #7
```

```
EXTRDDEFLIB7V                DS  CL72        DEFAULT NAMESTRING 7
EXTRDDEFLIB8N                DS  CL8         DEFAULT LIB NAME #8
EXTRDDEFLIB8V                DS  CL72        DEFAULT NAMESTRING 8
EXTRDDEFLIB9N                DS  CL8         DEFAULT LIB NAME #9
EXTRDDEFLIB9V                DS  CL72        DEFAULT NAMESTRING 9
EXTRDDEFLIB10N               DS  CL8         DEFAULT LIB NAME #10
EXTRDDEFLIB10V               DS  CL72        DEFAULT NAMESTRING 10
*
***********************CLASS 2*****************************************
           ORG EXTRDBEGIN
EXTRDCLASS2                  DS  0XL8        RETURNED FOR CLASS 2
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
*                                               (BACKUP SKIPS)
EXTRDIDCPU                   EQU 61,2,H      CURRENT CPU ID
EXTRDIDHZ                    EQU 62,2,H      A/C LINE FREQUENCY
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      HIGHEST DEVICE # IN CONFIG
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
*                                                CENTRAL DISKETTE
EXTRDIDCDISKET               EQU 66,2,H       DEVICE # OF SYSTEM'S
*                                             CENTRAL DISKETTE --
*                                             ALIAS FOR EXTRDCDISKET
EXTRDIDGENFLAG               EQU 75,4,A       MCBGENFLAGS CONTENTS
*                                             (GENEDITOR FLAGS)
EXTRDGENDMSTX                EQU X'04'        DMS/TX Supported
EXTRDGENDMSTXB               EQU 29           Bit displacement into
*                                             MCBGENFLAG word for
*                                             use with BTEST instr
*                                             on DMSTX support bit
EXTRDGENWP                   EQU X'02'        WP SUPPORTED ON SYSTEM
EXTRDGENWPB                  EQU 30           BIT DISPLACEMENT INTO
*                                             MCBGENFLAG WORD FOR
*                                             USE WITH BTEST INSTR
*                                             ON WP SUPPORT BIT
EXTRDGENMWAY                 EQU X'01'        MAILWAY SUPPORTED
EXTRDGENMWAYB                EQU 31           BIT DISPLACEMENT INTO
*                                             MCBGENFLAG WORD FOR
*                                             USE WITH BTEST INSTR
*                                             ON MAILWAY SUPPORT BIT
*
EXTRDGENOFFICE               EQU X'08'        Wang Office supported
EXTRDGENOFFICB               EQU 28           BIT DISPLACEMENT INTO
*                                             MCBGENFLAG WORD FOR
*                                             USE WITH BTEST INSTR
*                                             ON WANG OFFICE SUPPORT
*                                             BIT
*

EXTRDPRTFCLAS                EQU 76,1,C       Print file class
EXTRDSYSNAME                 EQU 77,16,C      System name
EXTRDSESMPORT                EQU 78,4,C       Session mgr port name
EXTRDFTMPORT                 EQU 79,4,C       File transfer mgr port
EXTRDTSKMPORT                EQU 80,4,C       Task manager port name
EXTRDSYSTPORT                EQU 81,4,C       System task port name
EXTRDPRTTPORT                EQU 82,4,C       Printer task port name
EXTRDSHRPORT                 EQU 83,4,C       Sharer port name
EXTRDSEG2SIZE                EQU 84,2,H       DEFAULT USER MOD. AREA SIZE
EXTRDSYSTEMID                EQU 85,8,C       System Wangnet ID
EXTRDNETCNFG                 EQU 86,8,C       Wangnet config file
EXTRDSHRADDR                 EQU 87,4,A       Addr of shared area
EXTRDSHRSIZE                 EQU 88,4,A       Size of shared area
EXTRDIDDATEFMT               EQU 91,1,C       Date format
EXTRDIDUIOSWAGE              EQU 92,2,H       System autologoff time
EXTRDIDSDMSTXTO              EQU 94,1,H       SHARER's DMSTX timeout
EXTRDIDHIADR                 EQU 96,4,A       Physical memory size+1
EXTRDOPERMSG                 EQU 98,80,C      Oper broadcast message
EXTRDIDOPERMSG               EQU 98,80,C      OPER BROADCAST
EXTRDIDUSERMSG               EQU 99,80,C      Oper to User message
*
```

```
***********************************************************************
* 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
EXTRDIDURDFLGS               EQU 64,4,B      USER'S 'READ' ACCESS
EXTRDIDUWTFLGS               EQU 65,4,B      USER'S 'WRITE' ACCESS
EXTRDIDSEG2BUF               EQU 17,2,H      NUMBER OF USER MODIFIABLE
*                                               AREA '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
*                                               CONSTRUCTED FROM USER ID
*                                               OR BG TASK #
EXTRDIDSPOOLIB               EQU 32,8,C      SPOOL LIBRARY NAME
*                                               CONSTRUCTED FROM USER ID
*                                               OR BG TASK #
EXTRDIDJOBNAME               EQU 71,8,C      NAME OF BACKGROUND JOB
EXTRDIDSEG2SIZE              EQU 33,4,F      LENGTH OF USER MOD. AREA (BYTES)
EXTRDIDSTATIC                EQU 34,4,A      ADDRESS OF START OF STATIC
*                                               AREAS (R14 AT PROGRAM
*                                               INVOCATION)
EXTRDIDLOGPTR                EQU 89,4,A      PROCEDURE LOG FILE
*                                            CONTROL BLOCK PTR
EXTRDIDTASKPRNT              EQU 95,4,A      PARENT'S TASK #
*
```

```
************************************************************************
* 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
*                                               ('R' OR 'H')
EXTRDIDJOBCLASS              EQU 69,1,C      DEFAULT JOB CLASS
*                                               ('A' TO 'Z')
EXTRDIDJOBLIMIT              EQU 70,4,F      DEFAULT JOB CPU TIME
*                                               LIMIT (SECONDS)
EXTRDIDSPOOLSYS              EQU 90,8,C      DEFAULT SYSTEM FOR
*                                               PRINT ROUTING
EXTRDIOPERMSGS               EQU 97,1,C      Status of Help on
*                                               Opr Msgs flag
*
************************************************************************
* RUN STATISTICS
************************************************************************
*
EXTRDIDWSIO                  EQU 41,4,F      COUNT OF WORKSTATION I/O'S
*                                               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
EXTRDIDVOICEIO               EQU 100,4,F     COUNT OF VOICE 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:
           ORG EXTRDBEGIN
EXTRDDEVCLASS                DS  HL1         DEVICE CLASS:
EXTRDDEVCLASSWS              EQU 1              WORKSTATION
EXTRDDEVCLASSMT              EQU 2              MAGNETIC TAPE
EXTRDDEVCLASSDK              EQU 3              DISK

EXTRDDEVCLASSPR              EQU 4              PRINTER
EXTRDDEVCLASSTC              EQU 5              TELECOMMUNICATIONS
*EXTRD&SUFFIX.DEVCLASSFP      EQU 6          FRONT END PROCESSOR
EXTRDDEVCLASSVC              EQU 6              VOICE DEVICE
EXTRDTYPE                    DS  HL1         DEVICE TYPE:
EXTRDTYPE2246P               EQU 017         2246P   WORKSTATION
EXTRDTYPE2246S               EQU 018         2246S   WORKSTATION
EXTRDTYPE2246R               EQU 019         2246R   WORKSTATION
EXTRDTYPE2246C               EQU 020         2246C   WORKSTATION
EXTRDTYPE2246K               EQU 021         2246K   WORKSTATION
EXTRDTYPE2266C               EQU 022         ARCHIVER C W/S
EXTRDTYPE2266S               EQU 023         ARCHIVER S W/S
EXTRDTYPE2246SI              EQU 024         IDEOGRAPHIC S W/S
EXTRDTYP2246CIJ              EQU 025         Ideographic J C W/S
EXTRDTYPE2256C               EQU 026         64K C W/S
EXTRDTYPE2276C               EQU 027         ARCHIVER C 64K W/S
EXTRDTYPE2246O               EQU 028         OKIDATA WORKSTATION
EXTRDTYPE2246CI              EQU 029         IDEOGRAPHIC C W/S
EXTRDTYP2246SIK              EQU 030         IDEOGRAPHIC/K S W/S
EXTRDTYPE2246RK              EQU 031         REMOTE KATAKANA W/S
```

```
EXTRDTYP2246SIJ              EQU 032         IDEOGRAPHIC/J S W/S
EXTRDTYPE2246CD              EQU 033         2246SCD CASH DRAWER WS
*
EXTRDTYPE2209V               EQU 034         2209V MAG TAPE
*                                               (1600 BPI)
EXTRDTYPE2209V2              EQU 035         2209V-2 MAG TAPE
*                                               (800/1600 BPI)
EXTRDTYPE2209V3              EQU 036         2209V-3 7-TRACK MAG TAPE
EXTRDTYPE2219V1              EQU 037         2219V-1 9-TRACK 75 IPS
*                                            TAPE (1600/6250 BPI)
EXTRDTYPE2219V2              EQU 038         2219V-2 9-TRACK 125 IPS
*                                            TAPE (1600/6250 BPI)
EXTRDTYPE2219V3              EQU 039         2219V-3 9-TRACK 75 IPS
*                                            TAPE (800/1600/6250)
EXTRDTYPE2219V4              EQU 040         2219V-4 9-TRACK 125 IPS
*                                            TAPE (800/1600/6250)
EXTRDTYPE2246S1              EQU 041         029 STANDARD KEYPAD WS
EXTRDTYPE2246S2              EQU 042         REVERSED NUMERIC KP WS
EXTRDTYPE2246S3              EQU 043         029, REV NUMERIC KP WS
EXTRDTYP2246SDB              EQU 044         BIZDIAL workstation
EXTRDTYPE2529V               EQU 045         2529V CARTRIDGE TAPE
EXTRDTYPE2509V               EQU 046         2509V SERIAL TAPE
*
EXTRDTYPE2260V               EQU 050         2260V  DISK (408CYL F/R)
EXTRDTYPE2265V1              EQU 051         2265V-1 DISK(823CYL REM)
EXTRDTYPE2265V2              EQU 052         2265V-2 DISK(823CYL REM)
EXTRDTYPE2270V               EQU 053         2270V   DISKETTE
*                                               (77CYL REM)
EXTRDTYPE2280V1              EQU 054         2280V-1 DISK(823CYL F/R)
EXTRDTYPE2280V2              EQU 055         2280V-2 DISK(823CYL F/R)

EXTRDTYPE2280V3              EQU 056         2280V-3 DISK(823CYL F/R)
EXTRDTYPE2270V1              EQU 057         2270V-1 DISKETTE
*                                            (HARD SECTORED)
EXTRDTYPE2270V2              EQU 058         2270V-2 DISKETTE
*                                            (SOFT SECTORED)
EXTRDTYPE2270V3              EQU 059         2270V-3 DISKETTE
*                                            (HARD OR SOFT SECTORED)
EXTRDTYPE9614                EQU 060         9614 FIXED DISK
*
EXTRDTYP2265V1A              EQU 061         2265V1A DUAL PORT
*                                               ( 75 MEG )
EXTRDTYP2265V2A              EQU 062         2265V2A DUAL PORT
*                                               ( 288 MEG )
EXTRDTYPE2270V4              EQU 063         2270V-4 Diskette
*                                            (soft sectored)
EXTRDTYPE2265V3              EQU 064         2265V-3 FIXED DISK
*                                               (622 MEG)
EXTRDTYPE2221V               EQU 065         2221V   PRINTER
*                                               (200CPS MAT)
```

```
EXTRDTYP2265V3A              EQU 066         2265V-3A FIXED DISK
*                                               (620 MEG)
EXTRDTYPE2231V2              EQU 067         2231V-2 PRINTER
*                                               (120CPS MAT)
EXTRDTYPE2263V1              EQU 069         2263V-1 PRINTER
*                                               (300LPM TR)
EXTRDTYPE2263V2              EQU 070         2263V-2 PRINTER
*                                               (600LPM TR)
EXTRDTYPE2281V               EQU 073         2281V   PRINTER (30CPS
*                                               DAISY WHEEL)
EXTRDTYPE2263V3              EQU 075         2263V-3 PRINTER
*                                               (430 LPM TR)
EXTRDTYPE2273V1              EQU 076         2273V-1 PRINTER
*                                               (REMOTE)
EXTRDTYPE2281WR              EQU 077         2281WR PARALLEL PRT
*                                              (REMOTE DAISY)
EXTRDTYP2281WCR              EQU 078         2281WCR PARALLEL PRT
*                                              (REMOTE DAISY)
EXTRDTYPE2233                EQU 082         2233 PARALLEL PRT
*                                              (REMOTE DOT MATRIX)
EXTRDTYPE2235                EQU 083         2235 PARALLEL PRT
*                                              (REMOTE DOT MATRIX)
EXTRDTYPE2233K               EQU 084         2233K REMOTE KAT MATR
*
EXTRDTYPE2235K               EQU 085         2235K REMOTE KAT MATR
*
EXTRDTYPETC2                 EQU 086         TC2 - BATCH TC DEVICE
*                                              (ON 22V56-1 IOP)
EXTRDTYPETC3                 EQU 087         TC3 - BATCH TC DEVICE
*                                              (ON 22V66-1 IOP)
EXTRDTYPETC4                 EQU 088         TC4 - BATCH TC DEVICE

*                                             (ON 25V76-1 ADAPTER)
*** Serial Printers
*
EXTRDTYPE5521                EQU 097         5521 PRINTER
*                                               (200 CPS MATRIX)
EXTRDTYPE55312               EQU 099         5531-2 PRINTER
*                                               (120 CPS MATRIX)
EXTRDTYPE5577                EQU 100         5577 PRINTER
*                                               (HIGH DENSITY)
EXTRDTYPELPS12               EQU 101         LPS-12 LASER PRINTER
*
EXTRDTYPE5570                EQU 102         5570 PRINTER
*                                               (600 LPM TR)
EXTRDTYPEDW20                EQU 104         DW20 PRNTR
*                                               ( 20 CPS DAISY)
EXTRDTYPE6581W               EQU 105         6581W PRINTER
*                                               ( 30 CPS DAISY)
EXTRDTYPE5571                EQU 107         5571 PRINTER
*                                               (430 LPM TR)
```

```
EXTRDTYPE6581WC                EQU 108          6581-WC WIDE PRINTER
*                                                  ( 40 CPS DAISY )
EXTRDTYPE5573                  EQU 109          5573 PRINTER
*                                                  (300 LPM BAND)
EXTRDTYPE5574                  EQU 110          5574 PRINTER
*                                                  (600 LPM BAND)
EXTRDTYPE5521K                 EQU 111          5521K KATAKANA PRINTER
*                                                  (200 CPS MATRIX)
EXTRDTYPE55312K                EQU 112          5531-2K KATAKANA PRT
*                                                  (120 CPS MATRIX)
EXTRDTYPE5548Z                 EQU 113          5548Z TYPESETTER
*
EXTRDTYPEIP41D                 EQU 114          INTELLIGENT IMAGE PRT
*
EXTRDTYPE5521I                 EQU 115          IDEOGRAPHIC MAT   PRT
*
EXTRDTYPE5581WD                EQU 116          DUAL-HEAD DAISY PRT
*
EXTRDTYPE5521IK                EQU 118          IDEOGRAPHIC/K MAT PRT
*
EXTRDTYPE5535                  EQU 119          180 CPS MAT PRT
*
EXTRDTYPEOK555                 EQU 120          OKIDATA MATRIX PRT
*
EXTRDTYPE5575                  EQU 121          HI-SPEED BAND PRINTER
*
EXTRDTYPE5590                  EQU 122          900 LPM ARABIC DRUM PRT
*
EXTRDTYPEBIZDL                 EQU 125          BIZDIAL AUTODIAL MODEM
*
EXTRDTYPE5533                  EQU 126          100 CPS MATRIX PRINTER

*
EXTRDTYPE5535K                 EQU 127          180 CPS Katakana Matrx
*
EXTRDTYPE5533K                 EQU 128          100 CPS Katakana Matrx
*
EXTRDTYPECIU                   EQU 129          CIU PROCESSOR
*
EXTRDTYPETCB1                  EQU 130          TCB1 DEVICE
*
EXTRDTYPE5556                  EQU 131          5556 OIS WORKSTATION
*
EXTYRDTYPETCB3                 EQU 132          TCB3 Device
*
EXTRDTYPE4210UK                EQU 133          UNIVERSAL KEYPAD WS
*
EXTRDTYPE4210BK                EQU 134          BANKING KEYPAD WS
*
EXTRDTYPES400                  EQU 135          VALIDATION PRINTER
*
```

```
EXTRDTYPEEXPWS      EQU 150    EXPERIMENTAL WS
*
EXTRDTYPEXPRTW      EQU 151    EXP PRINTER (NO DP)
*
EXTRDTYPEXPPRT      EQU 152    EXP PRINTER
*
EXTRDTYPEMWS0       EQU 153    Multitask W/S, port 0
*
EXTRDTYPEMWS1       EQU 154    Multitask W/S, port 1
*
EXTRDTYPEMWS2       EQU 155    Multitask W/S, port 2
*
EXTRDTYPEMWS3       EQU 156    Multitask W/S, port 3
*
EXTRDTYPEMWS4       EQU 157    Multitask W/S, port 4
*
EXTRDTYPEMWS5       EQU 158    Multitask W/S, port 5
*
EXTRDTYPEMWS6       EQU 159    Multitask W/S, port 6
*
EXTRDTYPEMWS7       EQU 160    Multitask W/S, port 7
*
EXTRDTYPE9614X      EQU 161    14-inch fixed disk
*                              (MAXTFR=2K)
*
EXTRDTYPEQ2040      EQU 162    8-inch fixed disk
*
EXTRDTYP6300GM2     EQU 169    Graphic Mono WP W/S
*
EXTRDTYPEIDD01      EQU 170    Ideo RAM Dictionary
*

EXTRDTYP2246SID     EQU 171    Ideo Serial Workstation
*
EXTRDTYP2246SJD     EQU 172    Ideo Serial Workstation
*
EXTRDTYPETPI1       EQU 173    Ideo Toshiba Prtr, TPI-1
*
EXTRDTYPENETMUX     EQU 174    Network Multiplexer
*
EXTRDTYPE6300GM     EQU 175    Graphic Mono WP W/S
*
EXTRDTYPE4200A      EQU 176    Wangnet Audio WS
*
EXTRDTYPE4300       EQU 177    Wangnet Combined WS
*
EXTRDTYPE4300A      EQU 178    Wangnet Audio Ws
*
EXTRDTYPE5537       EQU 179    Matrix Prtr (400 cps)
*
```

```
EXTRDTYPEDW55                EQU 180          Daisy Prtr (55 cps)
*
EXTRDTYPE4220R               EQU 181          Bisync Remote WS
*
EXTRDTYP4210WM               EQU 182          Monochrome DP WS
*
EXTRDTYPE2246DE              EQU 184          Data Entry Terminal
*
EXTRDTYPE2220                EQU 185          8-in, 75-MB fixed disk
*
EXTRDTYPE4210NL              EQU 186          32-LINE WS
*
EXTRDTYPE4230                EQU 187          Monochrome Combined WS
*
EXTRDTYPED2257               EQU 188          160-MB 8-inch fixed disk
*
EXTRDTYPE2270V5              EQU 189          5-1/4" console diskette
*
EXTRDTYPE2230                EQU 190          5-1/4" fixed winch disk
*
EXTRDTYPE4230A               EQU 191          English/Arabic WS
*
EXTRDTYPE4205                EQU 192          DP only 32k WS
***
EXTRDTYPE4240                EQU 193          BIT MAPPED WS
***
EXTRDTYPE4245                EQU 194          COLOR BIT MAPPED WS
***
EXTRDTYPE4235                EQU 195          COLOR CHARACTER WS
***
EXTRDTYPE4280                EQU 196          8086 BIT MAPPED WS
***

EXTRDTYP4210WM2              EQU 197          4210 GRAPHICS SUB-WS
***
EXTRDTYPE4245G2              EQU 198          4245 GRAPHICS SUB-WS
***
EXTRDTYPEPC                  EQU 199          PROF. COMPUTER WS
***
EXTRDTYPEPC2                 EQU 200          PC GRAPHICS TWIN WS
***
EXTRDTYPEPIC                 EQU 201          PROF IMAGE COMPUTER WS
***
EXTRDTYPEPIC2                EQU 202          PIC GRAPHICS TWIN WS
***
EXTRDTYPETC                  EQU 081          BATCH TC DEVICE
EXTRDTYPEIOMP1               EQU 203          VOICE - IOMP1 DEVICE
*
EXTRDDEVUSAGE                DS  CL2          DEVICE USAGE:
EXTRDDEVUEX                  EQU C'EX'           EXCLUSIVE USE
```

```
EXTRDDEVUSH                 EQU C'SH'          SHARED USE
EXTRDDEVUDT                 EQU C'DT'          DETACHED
EXTRDDEVUSER                DS  AL4         TASK IDENTIFIER OF
*                                              CURRENT DEVICE OWNER,
*                                              -1 IF NONE
EXTRDDEVREM                 DS  CL6         VOLSER OF REMOVABLE VOLUME
*                                           DEFINED ONLY FOR DISK AND
*                                           TAPE. CL6' ' IF NOTHING
*                                           MOUNTED.
EXTRDDEVFIXED               DS  CL6         VOLSER OF FIXED VOLUME
*                                           DEFINED ONLY FOR DISK.
*                                           CL6' ' IF NOTHING MOUNTED.
EXTRDDEVSPEC                DS  CL1         DEVICE SPECS FOR TAPES
*
EXTRDDEVST                  EQU C'T'        DRIVE SUPPORTS 3 DENSITIES
EXTRDDEVSD                  EQU C'D'        DRIVE SUPPORTS 2 DENSITIES
*
EXTRDDEVSPEC2               DS  CL1         MORE DEVICE SPECS
*
EXTRDDEVS2G                 EQU C'G'        DRIVE SUPPORTS 6250 BPI
*                                           DENSITY (GCR MODE)
EXTRDDEVPDA                 DS  H           Physical Device Addr
*XTRD&SUFFIX.DEVSPARE       DS  CL2         (UNUSED)
**********************************************************************
*
EXTRDIDVOLUME               EQU 53,32,B
*   INPUT  = VOLUME SERIAL NUMBER (6 BYTES)
*   OUTPUT AS FOLLOWS:
           ORG EXTRDBEGIN
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.
EXTRDVOLTYPE                DS  CL1         VOLUME TYPE:
EXTRDVOLTYPER               EQU C'R'           REMOVABLE
EXTRDVOLTYPEF               EQU C'F'           FIXED
EXTRDVOLLABEL               DS  CL2         LABEL TYPE:
EXTRDVOLLABSL               EQU C'SL'          STANDARD LABEL
EXTRDVOLLABNL               EQU C'NL'          NO LABEL
EXTRDVOLUSAGE               DS  CL2         VOLUME USAGE:
EXTRDVOLUSH                 EQU C'SH'           SHARED USE
EXTRDVOLURR                 EQU C'RR'           RESTRICTED..
*                                                   ..REMOVAL
EXTRDVOLUPR                 EQU C'PR'           PROTECTED USE
EXTRDVOLUEX                 EQU C'EX'           EXCLUSIVE USE
EXTRDVOLUSER                DS  AL4         TASK IDENTIFIER OF VOLUME
*                                              MOUNTER, -1 IF NONE
EXTRDVOLBC                  DS  HL2         BLOCKS PER CYLINDER
```

```
EXTRDVOLMAXTFR               DS  HL2         MAXIMUM TRANSFER IN BYTES
EXTRDVOLCV                   DS  HL2         CYLINDERS PER VOLUME
EXTRDVOLCVP                  DS  HL2         CYLINDERS PER PHYSICAL
*                                               VOLUME, INCLUDING BAD
*                                               AND UNUSED BLOCKS
EXTRDVOLOCNT                 DS  HL2         NUMBER OF FILES OPEN
EXTRDVOLSECT                 DS  CL1         SECTOR TYPE:
*                                              -- DISKETTE ONLY --
EXTRDVOLSECTS                EQU C'S'            SOFT SECTOR
EXTRDVOLSECTH                EQU C'H'            HARD SECTOR
EXTRDVOLADDR                 DS  CL1         ADDRESSING IN EFFECT:
*                                              -- DISKETTE ONLY --
EXTRDVOLADDRN                EQU C'N'            NONSTANDARD
EXTRDVOLADDRS                EQU C'S'            STANDARD
EXTRDVOLTOL                  DS  CL2         FAULT TOLERANCE LEVEL:
EXTRDVOLTOLNO                EQU C'  '         NO TOLERANCE
EXTRDVOLTOLCT                EQU C'CT'         CRASH TOLERANT
EXTRDVOLTOLMT                EQU C'MT'         MEDIA TOLERANT
EXTRDVOLPAGE                 DS  CL1         PAGING FILE ELIGIBILITY
EXTRDVOLPAGEY                EQU C'Y'          PAGING FILES ALLOWED
EXTRDVOLPAGEN                EQU C'N'          PAGING NOT ALLOWED
EXTRDVOLSPOOL                DS  CL1         SPOOL FILE ELIGIBILITY
EXTRDVOLSPOOLY               EQU C'Y'        SPOOL FILES ALLOWED
EXTRDVOLSPOOLN               EQU C'N'        SPOOL FILES NOT ALLOWED
EXTRDVOLWORK                 DS  CL1         WORK FILE ELIGIBILITY
EXTRDVOLWORKY                EQU C'Y'        WORK FILES ALLOWED
EXTRDVOLWORKN                EQU C'N'        WORK FILES NOT ALLOWED
EXTRDVOLSECURE               DS  CL1         SECURE VOLUME
EXTRDVOLSECUREY              EQU C'Y'        VOLUME IS SECURE
EXTRDVOLSECUREN              EQU C'N'        VOLUME IS NOT SECURE
EXTRDVOLCVD                  DS  HL2         CYLINDERS PER DIAGNOSTIC
*                                               VOLUME. INCLUDES ALL
*                                               OF EXTRDVOLCVP PLUS
*                                               DIAGNOSTIC CYLS

EXTRDXLMTOPEN                DS  XL1         XTNT LIMIT FOR FILE
*                                              CREATION
EXTRDXLMTTOTL                DS  XL1         TOTAL EXTENT LIMIT
*
***********************************************************************
*
EXTRDIDOTASK                 EQU 54,120,B
*   INPUT  = TASK IDENTIFIER (4 BYTES)
*   OUTPUT AS FOLLOWS:
           ORG EXTRDBEGIN
EXTRDOTASKWS                 DS  AL1         WORKSTATION DEVICE NUMBER
*                                               OF TASK SPECIFIED,
*                                               OR -1 IF NOT FOREGROUND
*                                               TASK
```

```
EXTRDOTASKUID       DS  CL3         CURRENT USER ID FOR TASK
*                                      SPECIFIED, OR BLANK
EXTRDOTASKNAME      DS  CL24        CURRENT USER NAME FOR TASK
*                                      SPECIFIED, OR BLANK
EXTRDOTASKTYPE      DS  CL2         TASK TYPE -
*                                        SEE EXTRDIDTASKTYPE
EXTRDOTASKSTAT      DS  CL18        STATUS OF TASK
EXTRDOTASKPROGI     DS  CL8         NAME OF INITIAL PROGRAM
*                                   RUN
EXTRDOTASKPROGC     DS  CL8         NAME OF CURRENT PROGRAM
*                                   RUN
EXTRDOTASKIDATE     DS  PL4         INITIAL PROGRAM START
*                                   DATE
EXTRDOTASKITIME     DS  FL4         INITIAL PROGRAM START
*                                   TIME
EXTRDOTASKETIME     DS  FL4         ELAPSED TIME SINCE CP
*                                   INITIATION, IN HUNDREDTHS
*                                   OF SECONDS
EXTRDOTASKPTIME     DS  FL4         PROCESSOR TIME SINCE CP
*                                   INITIATION, IN HUNDREDTHS
*                                   OF SECONDS
EXTRDOTASKWSIO      DS  FL4         COUNT OF WORKSTATION
*                                   I/O SINCE INITIAL PROGRAM
*                                   RUN
EXTRDOTASKDSKIO     DS  FL4         COUNT OF DISK I/O SINCE
*                                   INITIAL PROGRAM RUN
EXTRDOTASKTAPIO     DS  FL4         COUNT OF TAPE I/O SINCE
*                                   INITIAL PROGRAM RUN
EXTRDOTASKPRTIO     DS  FL4         COUNT OF PRINTER I/O SINCE
*                                   INITIAL PROGRAM RUN
EXTRDOTASKVCIO      DS  FL4         COUNT OF VOICE I/O FRM
*                                   INITIAL PROGRAM RUN
EXTRDOTASKOTIO      DS  FL4         COUNT OF OTHER I/O SINCE
*                                   INITIAL PROGRAM RUN
EXTRDOTASKPICNT     DS  FL4         PROGRAM PAGEIN COUNT
EXTRDOTASKPOCNT     DS  FL4         PROGRAM PAGEOUT COUNT

EXTRDOTASKSICNT     DS  FL4         SYSTEM PAGEIN COUNT
EXTRDOTASKSOCNT     DS  FL4         SYSTEM PAGEOUT COUNT
EXTRDOTASKSPARE     DS  BL4         RESERVED
*
```

```
***********************************************************************
*
EXTRDIDTAPEVOL               EQU 55,20,B
*   INPUT  = VOLUME SERIAL NUMBER (6 BYTES)
*   OUTPUT AS FOLLOWS:
           ORG EXTRDBEGIN
EXTRDTAPEDEV                 DS  AL1         DEVICE NUMBER, OR -1 IF
*                                               VOLUME NOT MOUNTED
EXTRDTAPESPAR1               DS  BL1         (UNUSED)
*
* NOTE: EXTRDTAPELABEL,
*       EXTRDTAPEUSAGE, AND EXTRDTAPEUSER ARE ALL BLANK
*       IF NO TAPE MOUNTED.
EXTRDTAPEDEN                 DS  HL2         TAPE DENSITY IN BINARY
*                                               BPI (556,800 OR 1600)
EXTRDTAPELABEL               DS  CL2         LABEL TYPE:
EXTRDTAPELABAL               EQU C'AL'          ANSI LABEL
EXTRDTAPELABNL               EQU C'NL'          NO LABEL
EXTRDTAPELABIL               EQU C'IL'          IBM LABEL
EXTRDTAPEUSAGE               DS  CL2         VOLUME USAGE:
EXTRDTAPEUSH                 EQU C'SH'          SHARED USE
EXTRDTAPEUEX                 EQU C'EX'          EXCLUSIVE USE
EXTRDTAPEUSER                DS  AL4         TASK IDENTIFIER OF VOLUME
*                                               MOUNTER, -1 IF NONE
EXTRDTAPEFSEQ                DS  HL2         FILE SEQUENCE NUMBER
EXTRDTAPESPAR2               DS  BL6         (UNUSED)
*
***********************************************************************
*
EXTRDIDDEVLIST               EQU 59,3,B
*   INPUT  = DEVICE CLASS, AS IN EXTRDDEVCLASS (1 BYTE)
*   OUTPUT AS FOLLOWS:
           ORG EXTRDBEGIN
EXTRDDLTOT                   DS  HL1         TOTAL NUMBER OF DEVICES IN
*                                               SPECIFIED CLASS
EXTRDDLNUM                   DS  HL1         NUMBER OF DEVICES
*                                               ADDRESSES SUPPLIED
EXTRDDLIST                   DS  0X          DEVICE LIST
EXTRDDLENTRY                 DS  AL1         DEVICE ADDRESS, OR X'FF'
*                                               IF NO MORE DEVICES
***********************************************************************
*
*                  TC RELATED INFORMATION
EXTRDIDDLPNAME               EQU 72,38,B
*   INPUT = DLPNAME (4 BYTE CHAR. STRING)
*   OUTPUT AS FOLLOWS:

*
```

```
         ORG   EXTRDBEGIN
EXTRDDLPDEVMAP               DS   XL4        BITMAP OF DEVS ON DLP
EXTRDDLPDEV#1                DS   XL2        1ST DEV ON DLP
EXTRDDLPTYPE                 DS   XL1        No. from PB or UCB
EXTRDDLPLINECNT              DS   XL1        # OF LINES CONTROLLABLE
*                                            BY DLP
EXTRDMCSTATUS                DS   XL1        MICROCODE FILE STATUS
*                                            0 - IF STOPPED
*                                            HI BIT ON IF LOADED
EXTRDDLPKIND                 DS   XL1        Periph./Device Proc.
EXTRDDLPKINDPP               EQU 0           Peripheral Processor
EXTRDDLPKINDDP               EQU 1           Device Processor
EXTRDDLPSPARE                DS   XL2        RESERVED FOR FUTURE
EXTRDMCFILE                  DS   XL8        MICROCODE FILE NAME
EXTRDMCLIB                   DS   XL8        MICROCODE LIB  NAME
EXTRDMCVOL                   DS   XL6        VOLUME NAME FOR MCFILE
*
EXTRDDLPRSRV                 DS   XL1        RESERVATION STATUS-DLP
*                                            HI BIT ON IF RESERVED
EXTRDDLPTASK#                DS   XL3        RESERVING TASK #
         ORG   ,
********************************************************************
*
EXTRDIDDLPDEV#               EQU 73,8,B
*   INPUT = DEVICE NUMBER (2 BYTES)
*   OUTPUT AS FOLLOWS:
*
         ORG   EXTRDBEGIN
EXTRDDEVSTATUS               DS   XL1        DEV RESERVATION STATUS
EXTRDDEVRSRV                 EQU X'40'       DEVICE RESERVED
EXTRDDEVOPEN                 EQU X'80'       DEVICE OPEN
EXTRDDEVTASK#                DS   XL3        RESERVING TASK #
EXTRDDEVDLPNAME              DS   XL4        DLPNAME FOR DEVICE
         ORG   ,
**********************************************************************
*
EXTRDIDVOLVCB                EQU 74,08,B
*   INPUT  = VOLUME SERIAL NUMBER (6 BYTES)
*   OUTPUT AS FOLLOWS:
           ORG EXTRDBEGIN
EXTRDVCBADDR                 DS   A           VCB ADDRESS, OR 0 IF
*                                             VOLUME NOT MOUNTED
                             DS   A           (UNUSED)
         ORG   ,
**********************************************************************
**********************************************************************
*
EXTRDIDDEFLIB                EQU 92,72,C
*   INPUT  = DEFAULT LIBRARY NAME (8 BYTES)
*   OUTPUT AS FOLLOWS:
           ORG EXTRDBEGIN
EXTRDDEFLIB                  DS  CL72        NAMESTRING CORRESPON-
*                                            DING TO USER-SUPPLIED
*                                            DEFAULT LIB NAME
         ORG   ,
***********************************************************************
*
*                  DEVICE CLUSTER INFORMATION
EXTRDIDCLUSTER               EQU 67,16,B
*   INPUT  = DEVICE NUMBER (2 BYTES)
*   OUTPUT AS FOLLOWS:
           ORG EXTRDBEGIN
EXTRDADISKET                 DS  HL2         DEVICE # OF ASSOCIATED
*                                               ARCHIVER DISKETTE,
*                                               OR ZERO IF NONE
EXTRDNEXTDEV                 DS  HL2         DEVICE # OF NEXT DEVICE
*                                               ON CLUSTER OR ZERO
*                                               IF NONE
EXTRDPORT                    DS  XL1         PORT ON IOP
EXTRDCHNL                    DS  XL1         CHANNEL ON BROADBAND
EXTRDASA                     DS  XL1         SHORT ADDRESS
EXTRDDCC                     DS  XL1         DEVICE ON CLUSTER
EXTRDCOP                     DS  XL1         CLUSTER ON PORT
EXTRDSPARE                   DS  BL7         (UNUSED)
*
         ORG
EXTRDIDIDERR                 DS  XL1         USED TO GENERATE DUMMY
*                                            FOR INVALID ID
************************************************************************
*
*                      AUTOMATIC LOGOFF WS ENABLE
EXTRDIDALOGF                 EQU 93,1,C
*  INPUT =  DEVICE NUMBER  (1 BYTE)
*  OUTPUT = "Y"    WORKSTATION AUTOLOGOFF ENABLED   OR
*           "N"    WORKSTATION AUTOLOGOFF DISABLED
EXTRDALOGF                   DS  CL1

         CSECT
```

4.2.25 FDR1 - Describe File Descriptor Record 1 (FDR1)

Syntax

```
    FDR1     [NODSECT][,REG=expression][,SUFFIX=character]
```

Function

The file descriptor record (FDR1) describes the attributes of a file, including the first three extents of the file for sinlge volumes only. Every file on a volume (except the VTOC and volume label/IPL text area) has an FDR1 associated with it. FDR1s 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 2043rd and 2044th bytes of the block. FDR1 records are present only on the root volume of a volume set.

Parameter Definitions

NODSECT — Specification of NODSECT results in the FDR1 fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, the system generates a DSECT with the name FDR1 (plus the optional suffix).

REG — Provides for the optional specification of a register for which a USING statement for the FDR1 fields is generated.

SUFFIX — If provided, all labels are generated by the concatenation of the letters FDR1, the user-provided SUFFIX (one ASCII character in length), and the field name.

Structure

FDR1

BEGIN |

	BYTE 0	BYTE 1	BYTE 2	BYTE 3	
+0	FORMAT	XTNTCOUNT	ORG	FLAGS	
+4	X1PTR		FILENAME		
+8					
+C			MOREFLAGS	CREDATE	+E ↕ FILESECTION
+10			MODDATE		
+14		EXPDATE			
+18	FPCLASS	CREATOR			
+1C	BLKSIZE		SECEXT		
+20	X1STRT			X1END	
+24			X2STRT		
+28		X2END			
+2C	X3STRT			X3END	
+30					

ORG - For files on volume sets:

	BYTE 0	BYTE 1	BYTE 2	BYTE 3	
+20	CHAIN3				
+24	#XTNT				
+28	SPARE0				
+2C	NBLKS				
+30	NSEGS		SPARE2		
+34	NRECS				
+38	RECSIZE		SPARE3	EBLK	
+3C			EREC		
+40	SPARE4				
+44					
+48					
+4C	CHAIN				LENGTH = 50

For word processing files only:

FDR1

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+30			WPBLKSIZE	WPBLS

For program files only:

		BYTE 0	BYTE 1	BYTE 2	BYTE 3
ACFLAGS	+40	WTFLAGS			
	+44	RDFLAGS			
	+48	EXFLAGS			

For indexed files only:

FDR1

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+30			PKI	PKD
+34				
+38			ALTCNT	
+3C				
+40	KEYPOS		KEYSIZE	HXBLK
+44			DABLK	
+48		PTRD		
+4C	CHAIN			

Example

```
          FDR1
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 5.03.03 (UPDATED FOR ADMS REMOVAL)
*
```

```
FDR1BEGIN                  DS  0F
FDR1FORMAT                 DS  CL1         FORMAT OF FDR (ASCII '1')
*                                  ('N' FOR FDR RECORD NOT IN USE)
FDR1INUSE                  EQU C'1'        FDR1 IN USE
FDR1NOTUSED                EQU C'N'        FDR1 NOT IN USE
FDR1XTNTCOUNT              DS  BL1         COUNT OF EXTENTS IN USE
FDR1ORG                    DS  BL1         FILE ORGANIZATION
FDR1ORGCONSEC              EQU X'01'       CONSECUTIVE ORGANIZATION
FDR1ORGINDEXED             EQU X'02'       INDEXED ORGANIZATION
FDR1ORGWP                  EQU X'04'       WORD PROCESSING FILE
FDR1ORGREL                         EQU        X'08'               Relative
Organization
FDR1ORGPLOG                EQU X'10'       FILE PROLOGUE PRESENT
FDR1ORGVLEN                EQU X'20'       VARIABLE-LENGTH RECORDS
FDR1ORGPRINT               EQU X'40'       PRINT FILE
FDR1ORGPROG                EQU X'80'       PROGRAM FILE
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
FDR1X1PTR                  DS  H           FDX1 BLOCK * 169 + FDX1
*                                          ITEM IN BLOCK (FROM 0)
FDR1FILENAME               DS  CL8         MEMBER NAME
FDR1FILESECTION            DS  CL1         VOLUME IN A MULTI-VOLUME
*                                          FILE (ALWAYS ASCII '1')
*
         ORG  FDR1FILESECTION
FDR1MOREFLAGS              DS  X           Additional Flags
FDR1TXALLOC                EQU X'80'       DMS/TX Blocks allocated
FDR1TXINUSE                EQU X'40'       DMS/TX Blocks in use
FDR1EXLOCKCL               EQU X'08'       SHARED FILE EXCLUSIVE
*                                          LOCK ON AT CLOSE TIME
*
FDR1CREDATE                DS  PL3         CREATION DATE (PACKED YYDDD+
FDR1MODDATE                DS  PL3         LAST MODIFICATION DATE
*                                          (PACKED YYDDD+)
FDR1EXPDATE                DS  PL3         EXPIRATION DATE (PACKED YYDD
FDR1FPCLASS                DS  CL1         FILE PROTECTION ACCESS-CLASS
FDR1CREATOR                DS  CL3         USER LOGON IDENTIFICATION OF
*                                          FILE CREATOR
```

```
FDR1BLKSIZE                DS  H          PHYSICAL BLOCK SIZE (2048)
FDR1SECEXT                 DS  H          NO. BLOCKS SECONDARY EXTENT
FDR1X1STRT                 DS  FL3        PRIMARY EXTENT START BLOCK
FDR1X1END                  DS  FL3        PRIMARY EXTENT END BLOCK + 1
FDR1X2STRT                 DS  FL3        2ND EXTENT START
FDR1X2END                  DS  FL3        2ND EXTENT END
FDR1X3STRT                 DS  FL3        3RD EXTENT START
FDR1X3END                  DS  FL3        3RD EXTENT END
* THE FOLLOWING OVERLAY ONLY FOR FILES ON VOLUME SETS
                     ORG   FDR1X1STRT
FDR1CHAIN3                DS  F (HL1,FL3) ADDRESS OF A FORMAT 3 FDR
*                              WHICH IS FOR THIS FILE'S BITMAP INFO
FDR1#XTNT                  DS  FL4        TOTAL NUMBER OF EXTENTS
FDR1$SPARE0                DS  BL4        UNUSED
FDR1NBLKS                  DS  BL4        TOTAL NUMBER OF BLKS
FDR1NSEGS                  DS  BL2        TOTAL NUMBER OF SEGS
*                                         IF EXTENT LIMITS ARE NOT
*                                         SET, THE VALUE WILL BE 0
************************************************************
* ORGANIZATION-DEPENDENT SECTION:
************************************************************
FDR1SPARE2                 DS  BL2        (UNUSED FOR CONSECUTIVE
*                                         FILES)
FDR1NRECS                  DS  F          NUMBER OF DATA RECORDS

FDR1RECSIZE                DS  H          LOGICAL RECORD SIZE
FDR1SPARE3                 DS  BL1        (UNUSED UNLESS
*                                         FDR1FLAGSALTX SET)
FDR1EBLK                   DS  FL3        LAST RECORD'S BLOCK WITHIN
*                                         FILE
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)
FDR1SPARE4                 DS  BL12       (UNUSED FOR CONSECUTIVE
*                                         FILES WHICH ARE NOT PROGRAM
*                                         FILES)
************************************************************
* FOR WORD PROCESSING FILES ONLY:
************************************************************
                           ORG FDR1SPARE2
FDR1WPBLKSIZE              DS  XL1        WP FILE BLOCK SIZE
FDR1WPBLS                  DS  XL1        BYTES IN LAST
*                                         SECTOR
```

```
*************************************************************
* FOR PROGRAM FILES ONLY:
*************************************************************
                           ORG FDR1SPARE4
FDR1ACFLAGS                DS  0BL12       ADDITIONAL ACCESS
*                                          PRIVILEGES:
FDR1WTFLAGS                DS  BL4         ADDITIONAL WRITE
*                                          PRIVILEGES
FDR1RDFLAGS                DS  BL4         ADDITIONAL READ
*                                          PRIVILEGES
FDR1EXFLAGS                DS  BL4         ADDITIONAL EXECUTE
*                                          PRIVILEGES
*************************************************************
* FOR INDEXED FILES ONLY (FILEORG X'02'):
*************************************************************
                           ORG FDR1SPARE2
FDR1PKI                    DS  HL1         PACKING FACTOR FOR INDEX
*                                          ITEMS
FDR1PKD                    DS  HL1         PACKING FACTOR FOR DATA
*                                          RECORDS
                           ORG FDR1SPARE3
FDR1ALTCNT                 DS  HL1         NUMBER OF ALTERNATE INDEX
*                                          STRUCTURES DEFINED IN THE
*                                          AXD1-BLOCK (UNUSED UNLESS
*                                          FDR1FLAGSALTX SET)
                           ORG FDR1SPARE4
FDR1KEYPOS                 DS  H           PRIMARY KEY POSITION IN
*                                          DATA RECORD
FDR1KEYSIZE                DS  HL1         PRIMARY KEY LENGTH IN BYTES
FDR1HXBLK                  DS  FL3         BLOCK-IN-FILE OF ROOT BLOCK

*                                          OF PRIMARY INDEX
FDR1DABLK                  DS  FL3         BLOCK-IN-FILE OF STARTING
*                                          BLOCK OF AVAILABLE-BLOCK
*                                          CHAIN
FDR1PTRD                   DS  FL3         FIRST DATA BLOCK IN FILE
*                                          (PRIMARY KEY SEQUENCE)
*************************************************************
* FDR CHAIN - IN ALL FDR RECORDS:
*************************************************************
FDR1CHAIN                  DS  F (HL1,FL3) ADDRESS OF A FORMAT 2 FDR
*                                          FOR THIS FILE'S ADDITIONAL
*                                          EXTENTS, THE ADDRESS IS IN THE
*                                          FORM:  (HL1) NUMBER STARTING
*                                          FROM 0 OF FDR IN 1-PAGE BLOCK
*                                          (FL3) BLOCK # IN VTOC FROM 0
FDR1END                    EQU *
FDR1LENGTH                 EQU FDR1END-FDR1BEGIN
FDR1CNT                    EQU 25          # OF FDR1 RECORDS PER BLOCK
         CSECT
```

4.2.26 FDR2 - Describe File Descriptor Record 2 (FDR2)

Syntax

```
    FDR2     [NODSECT][,REG=expression][,SUFFIX=character]
```

Function

This macro maps symbol names to an FDR2 record. The FDR2 record describes up to ten additional extents for a file for a single volume. For a set member, nine additional extents are described. FDR2 is chained from the file's FDR1 record and may be chained to another FDR2 record. For non-root volumes, FDR2 is chained from FDX.

Parameter Definitions

NODSECT — Specification of NODSECT results in the FDR2 fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, the system generates a DSECT with the name FDR2 (plus the optional suffix).

REG — Provides for the optional specification of a register for which a USING statement for the FDR2 fields is generated.

SUFFIX — If provided, all labels are generated by the concatenation of the letters FDR2, the user-provided SUFFIX (one ASCII character in length) and the field name.

Structure

```
                  BYTE 0     BYTE 1     BYTE 2     BYTE 3
                ________________________________________

       FDR2

BEGIN  |
          +0   | FORMAT   | SPARE1                        |
          +4   |                     | FILENAME           |
          +8   |                                          |
          +C   |                     | SPARE2             |
          +10  | X4STRT                         | X4END   |
          +14  |                     | X5TOX13            |
          +18  |                                          |
          +1C  |                                          |
          +20  |                                          |
          +24  |                                          |
          +28  |                                          |
          +2C  |                                          |
          +30  |                                          |
          +34  |                                          |
          +38  |                                          |
          +3C  |                                          |
          +40  |                                          |
          +44  |                                          |
          +48  |                                          |
          +4C  | CHAIN                                    |   LENGTH = 50

                  BYTE 0     BYTE 1     BYTE 2     BYTE 3
                ________________________________________
```

ORG - for multivolume files:

```
          +10  | VSID     | FSN                 | SPARE   |
          +14  | X1PTR               | X1STRT             |
          +18  |          | X1END                         |
          +1C  | X2TOX9                                   |
          +20  |                                          |
          +24  |                                          |
          +28  |                                          |
          +2C  |                                          |
          +30  |                                          |
          +34  |                                          |
          +38  |                                          |
          +3C  |                                          |
          +40  |                                          |
          +44  |                                          |
          +48  |                                          |
          +4C  | CHAIN                                    |   LENGTH = 50
```

Example

```
         FDR2
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 5.00
FDR2BEGIN                   DS  0F          FULLWORD ALIGNMENT
FDR2FORMAT                  DS  CL1         FORMAT (ASCII '2')
FDR2SPARE1                  DS  CL5         (UNUSED)
FDR2FILENAME                DS  CL8         FILE NAME AS IN FORMAT 1 FDR
FDR2SPARE2                  DS  CL2         (UNUSED)
FDR2X4STRT                  DS  FL3         EXTENT 4 (OR 14, 24, ETC.)
*                                   STARTING BLOCK ON VOLUME (FROM 0)
FDR2X4END                   DS  FL3         EXTENT 4 ENDING BLOCK ON VOL
FDR2XLEN                    EQU *-FDR2X4STRT           EXTENT LEN
FDR2X5TOX13                 DS  18FL3       EXTENT DEFINITIONS 5 TO 13
FDR2XCNT                 EQU (*-FDR2X4STRT)/FDR2XLEN
* FOLLOWING IS MULTI VOL FDR2 ENTRY
                   ORG      FDR2X4STRT             MULTI-VOL FILES
FDR2$VSID                   DS  BL1         CURRENT VSID
FDR2$FSN                    DS  BL2         FILE SEQUENCE NUMBER
FDR2$SPARE                  DS  BL1         (UNUSED)
FDR2$X1PTR                  DS  H           FDX1 BACK PTR
FDR2$X1STRT                 DS  FL3         EXTENT 1
FDR2$X1END                  DS  FL3         EXTENT 1 ENDING BLOCK
FDR2$XLEN                   EQU *-FDR2$X1STRT          ENTENT LEN
FDR2$X2TOX9                 DS  8FL6        EXTENT DEF 2 TO 9
FDR2$XCNT                EQU (*-FDR2$X1STRT)/FDR2$XLEN
FDR2CHAIN                   DS  F (BL1,FL3) CHAIN TO NEXT FORMAT 2 FDR
*                                   FOR ADDITIONAL EXTENTS
*                                   (SEE FDR1CHAIN)
FDR2END                     EQU *
FDR2LENGTH                  EQU FDR2END-FDR2BEGIN
```

4.2.27 FDR3 - Describe File Descriptor Record 3 (FDR3)

Syntax

```
    FDR3     [NODSECT][,REG=expression][,SUFFIX=character]
```

Function

This macro maps symbol names to an FDR3 record. The FDR3 record is present only for files on volume sets. It is chained from the file's FDR1 record and may be chained to another FDR3 record. The FDRS record is stored on the root volume.

Parameter Definitions

NODSECT — Specification of NODSECT results in the FDR3 fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, the system generates a DSECT with the name FDR3 (plus the optional suffix).

REG — Provides for the optional specification of a register for which a USING statement for the FDR3 fields is generated.

SUFFIX — If provided, all labels are generated by the concatenation of the letters FDR3, the user-provided SUFFIX (one ASCII character in length) and the field name.

Structure

```
                    BYTE 0     BYTE 1     BYTE 2     BYTE 3
                 ____________________________________________

        FDR3

BEGIN |
           +0    | FORMAT   | FILENAME                       |
           +4    |                                           |
           +8    |          | VSID      | FSN                |
           +C    | STRTBLK#                                  |
           +10   | 2TO9                                      |
           +14   |                                           |
           +18   |                                           |
           +1C   |                                           |
           +20   |                                           |
           +24   |                                           |
           +28   |                                           |
           +2C   |                                           |
           +30   |                                           |
           +34   |                                           |
           +38   |                                           |
           +3C   |                                           |
           +40   |                                           |
           +44   |                                           |
           +48   | SPARE1                                    |
           +4C   | CHAIN3                                    |   LENGTH = 50
```

Example

```
         FDR3
FDR3          DSECT
*
*        THE FORMAT 3 FILE DESCRIPTOR RECORD (FDR3) IS A NEW CONTROL
*        BLOCK (80 BYTES) FOR MULTIVOLUME FILES.  IT CONTAINS ENTRIES FOR
*        THE SEGMENT NUMBER, VSID, STARTING BLOCK IN FILE FOR SEGMENT,
*        AND THE CHAIN TO ANOTHER FDR3.
*        IT IS CHAINED FROM THE FILE'S FORMAT 1 FILE DESCRIPTOR
*        RECORD.
*
*
*
*
*
FDR3BEGIN                  DS  0F          FULLWORD ALIGNMENT
FDR3FORMAT                 DS  CL1         FORMAT (ASCII '3')
FDR3FILENAME               DS  CL8         COOKIE FILE NAME
FDR3VSID                   DS  BL1         VOLUME ID IN A SET
FDR3FSN                    DS  BL2         FILE SEGMENT NUMBER
FDR3STRTBLK#               DS  FL4         START BLOCK # IN A FILE
FDR32TO9                   DS  8BL7        2 TO 9 FILE SEGMENT ENTRIES
FDR3SPARE1                 DS  CL4         UNUSED IN FDR3
FDR3CHAIN3                 DS  F (BL1,FL3) CHAIN TO NEXT FDR3
FDR3END                    EQU *
FDR3LENGTH                 EQU FDR3END-FDR3BEGIN
         CSECT
```

4.2.28 FMTLIST - Generate Parameter Group Control Block

Syntax

```
[label] FMTLIST [LABELPFX='prefix'][,PREVIEW={YES}]
                                             {NO}

              {'keyword',({'default-value' }[,CHAR][,line-advance]
                          { absolute-length}[ INT ]
                                            [ NUM ]
                                            [ AN  ]
                                            [ HEX ]
                                            [UCHAR]
                                            [ ANL ]

                          [,space-advance])                            },

              {TEXT,('display-text'[,line-advance][,space-advance])    },
              {textname,('display-text'[,line-advance][,space-advance])},
```

Function

Generates a field format control block for use in a parameter group control list, which is required input to the GETPARM and the PUTPARM SVCs. The generated data structure is identical to the one generated by 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.

Parameter Definitions

LABELPFX — A character string in quotes which prefixes each keyword name, resulting in a string used to label each corresponding field format control block. The label is placed on the line-advance byte. Thus, for the keyword/receiving field format control block, the flag byte is at the location specified by this label +2, and the receiving field ('default-value') is at this location +12. This parameter is optional.

PREVIEW — If YES is specified, a simulated screen display is printed in the source listing (via comment level MNOTES) in the format specified by the macroinstruction parameters. If NO is specified, the display is not generated in the listing, and "CURRENT LINE LENGTH" messages are not generated. NO is the default.

keyword — A name of one to eight alphanumeric characters enclosed in single quotes, which identifies a specific parameter within the group. Keyword specification is mutually exclusive with specification of TEXT or text name.

default-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 default-value is mutually exclusive with specification of absolute-length.

absolute-length An absolute expression may be provided defining the length of the receiving field for this parameter. Specification of absolute-length is mutually exclusive with specification of default-value.

NOTE

Leading and trailing blanks are accepted in any of the formats listed below except the alphanumeric (AN) and the limited alphanumeric (ANL) formats, where only trailing blanks are accepted.

CHAR Any character is accepted as a valid response.

INT Only unsigned integers are accepted as a valid response.

NUM Numbers (with optional decimal point or leading sign or both) are accepted as a valid response.

AN Letters, including special characters #, @, and $, and numerals are accepted as a valid response. GETPARM converts lowercase letters to upper case.

HEX Only numerals and letters A-F are accepted as a valid response. Lowercase letters a-f are converted to uppercase.

UCHAR Any character is accepted as a valid response. Lowercase letters are converted to uppercase.

ANL Letters, including special characters #, @, and $, and numerals are accepted as a valid response. GETPARM converts lowercase letters to uppercase. The first character may not be a number.

line-advance The number of lines to advance before displaying the keyword and receiving field, or the embedded text. May be a value of 0 to 18 and if not specified, defaults to 1. If value is not 0, the keyword or text is displayed starting on the current line plus the line advance value. If 0, line advancing does not occur.

space-advance The number of spaces to advance within a line before displaying the keyword. May be specified as a value between 0 and 78. The default is 0. If specified or omitted, the value of space-advance plus 1 is the number of spaces that 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 alternate formats:

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

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

Regardless of how the space-advance is specified, an 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.

TEXT Indicates that embedded text is supplied in the following parameter.

Textname A nonquoted text name 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. Specification of TEXT or textname is mutually exclusive with specification of keyword.

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

Example

```
 FMT1    FMTLIST  LABELPFX='XXX',PREVIEW=YES,                         -
              TEXT1,('HEADING'),                                        -
              TEXT,('SUBHEADING'),                                      -
              'LIST',('NO',AN)
+FMT1    DC   HL1'0,3'                      PF KEY   &   FIELD COUNT
+        DC   AL1(1,0,-1,6)                 LINE / SPACE ADV, FLAGS, LGTH-1
+TEXT1   DC   C'HEADING'                    DISPLAYED TEXT
+             **  CURRENT LINE LENGTH IS 8  **
+        DC   AL1(1,0,-1,9)                 LINE / SPACE ADV, FLAGS, LGTH-1
+        DC   C'SUBHEADING'                 DISPLAYED TEXT
+             **  CURRENT LINE LENGTH IS 11  **
+XXXLIST DC   AL1(1,0,4,1)                  LINE / SPACE ADV, FLAGS, LGTH-1
+        DC   CL8'LIST',CL2'NO'             KEYWORD & DISPLAYED VALUE
+             **  CURRENT LINE LENGTH IS 14  **
+************************************************************************
+         1         2         3         4         5         6         7
+123456789012345678901234567890123456789012345678901234567890123456789012
+************************************************************************
+ HEADING
+ SUBHEADING
+ LIST     = NO
+
+************************************************************************
+         1         2         3         4         5         6         7
+123456789012345678901234567890123456789012345678901234567890123456789012
+************************************************************************
```

4.2.29 FREEALL - Free Resources (SVC 52)

Syntax

```
FREEALL [CANCEL={NO }][,ACK={NO }][{LEVELS={(Register)}}]
                {YES}       {YES}          {expression}

                                       {ALL={YES}          }
                                            {NO }
```

Function

Releases all resources in all shared DMS files opened by this task. Under advanced sharing may be used to hold or free extension rights including held records, keys, and files. Ends the current transactions for DMS/TX which commits updates and free locks. See the VS DMS/TX Reference for more information.

Parameter Definitions

CANCEL Optional parameter that specifies cancellation of the transaction on error. The default is NO.

ACK Optional parameter that specifies production of a message when an error is encountered. The default is NO.

LEVELS Number of levels to be committed. The value can be specified as a register containing the address of a fullword initialized with the level number or an expression. If not specified, the maximum positive integer is assumed.

ALL YES specifies that all levels are to be committed.

Stack On Input

```
         |                      |    Lower
         |                      |    Address
         |0     1     2     3   |
 0(SP)   |      |     |         |
         | (1)  | (2) |   (3)   |
         |      |     |         |
 4(SP)   |______|_____|_________|
         |                      |
         |                      |    Higher
         |______________________|    Address
         |      Preceding       |
         |      Stack Data      |
```

(1) Flag byte:

```
        Bit 0   1 = HOLD, 0 = RELEASE.
        Bit 1   1 = EXTENSION RIGHTS request.
        Bit 2   1 = RELEASE ALL and commit DMS/TX transaction.
        Bit 3   1 = TIME OUT in use.
        Bit 4   1 = Cancel on error.
        Bit 5   1 = Produce ACK GETPARM on error.
        Bit 6   Reserved for internal use, must be 0.
        Bit 7   Reserved for internal use, must be 0.
```

(2) Time out value in seconds from 0 - 255.

(3) Reserved, must be 0.

Stack On Output

```
             |                       |    Lower
             |                       |    Address
             |0     1     2     3    |
   0(SP)     |_______________________|
             | (1) Return Code       |
             |_______________________|
   4(SP)     |              |        |
             | (2) User ID  | (3)    |    Higher
             |______________|________|    Address
             |      Preceding        |
             |      Stack Data       |
```

(1) Return code
(2) User ID of user holding extension rights
(3) Unused

Output

A return code is issued in the topword of the stack, as follows:

Code	Definition
0	Success.
4	Timeout.
8	Invalid function sequence.
12	Request to HOLD or FREE with no shared files open.
16	System error - the sharer is not active or has run out of memory space.

Code	Definition
20	System error - before image journal (BIJ) error during end transaction.
24	Invalid function parameter.
28	Invalid subtransaction nesting.

Examples

```
    FREEALL CANCEL=YES
+   PUSHA 0,0
+   PUSHA 0,0
+   MVI   0(15),X'20'         FREE ALL SHARED RESOURCES
+   OI    0(15),X'08'         SET UP CANCEL-ON-ERROR CONDITION
+   SVC   52 (FREEALL)

    FREEALL ACK=YES
+   PUSHA 0,0
+   PUSHA 0,0
+   MVI   0(15),X'20'         FREE ALL SHARED RESOURCES
+   OI    0(15),X'04'         SET UP ACKNOWLEDGE-ERROR CONDITION
+   SVC   52 (FREEALL)
```

4.2.30 FREEBUF - Free Buffer Space (SVC 6)

Syntax

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

                [,LEVEL={(register)}]
                        {  address }
```

Function

To deallocate a buffer area allocated by the GETBUF SVC. The buffer area at the address specified by the BUFLOC parameter and for the length specified by the LENGTH parameter is made available for reallocation by GETBUF. The contents of this area should be considered unreliable after the FREEBUF has been issued.

Control register 2, the stack limit, may be modified.

Restrictions

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

Parameter Definitions

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. A LENGTH of 2048 is assumed if no LENGTH parameter is supplied.

LEVEL Process level at which to deallocate the block of memory. The default is the current process level. Available for privileged callers only. Specified as an address expression pointing to a 1-byte binary field containing the process level.

Stack On Input

```
                                            Lower
              |                      |      Address
              |______________________|
   0(SP)      |                      |
              | (1)  Buffer Address  |
              |______________________|
   4(SP)      |                      |
              | (2)  Buffer Size     |      Higher
              |______________________|      Address
              |      Preceding       |
              |     Stack Data       |
```

(1) Address of the buffer that is to be returned for use.

(2) The size of the buffer which must be a multiple of the page size (2048). This SVC is called by the GETHEAP/FREEHEAP SVCs. In the case of calls by the GETHEAP/FREEHEAP SVC, the high-order byte of this word 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.

Stack On Output

```
                                            Lower
              |                      |      Address
              |______________________|
   0(SP)      |                      |
              |     Return Code      |
              |______________________|
   4(SP)      |                      |
              |                      |      Higher
              |______________________|      Address
              |      Preceding       |
              |     Stack Data       |
```

Output

One of five return codes is issued in the topword on the stack. The return codes are as follows:

Code	Description
0	Buffer deallocated.
4	Invalid buffer address.
8	Invalid buffer length.
12	Process level requested is greater than caller's level.
16	Internal system error.

Example

```
          FREEBUF BUFLOC=OUTBUF,LENGTH=(R3)
+         PUSH     0,R3
+         PUSHA    0,OUTBUF
+         SVC      6    (FREEBUF)
```

4.2.31 FREEHEAP - Deallocate Heap Storage (SVC 57)

Syntax

```
[label] FREEHEAP    SIZE=(register),LINKLEV=address,

                    BUFLOC={(register)},POOLNAME={expression},
                           { address  }          { 'string' }

                    [ROOTLEV][,SEARCH][,DELETE][,LEVEL={  address }]
                                                       {(register)}
```

Function

To deallocate a block of memory which was previously allocated by the GETHEAP SVC. All block sizes, including zero, are legal but they are rounded to their nearest 8-byte multiple. This SVC calls FREEFBUF (SVC 6) to deallocate blocks which are greater than or equal to one page length in size. An entire subpool can also be deleted in a single FREEHEAP call, through the use of the DELETE flag. On UNLINK, all the subpools belonging to that link level are automatically deleted. FREEBUF may modify the value in control register 2.

Restrictions

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

Parameter Definitions

SIZE The size of the block to be deallocated. 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. The value can be specified as a register in parentheses that contains the start address of the buffer block in the low-order three bytes, or as an address expression pointing to a 4-byte field that 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.

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 1-byte field containing the link level in binary. Default is 0 (i.e., current link level).

LEVEL Process level at which to deallocate the block of memory. The default is the current process level. Available for privileged callers only. Specified as an address expression pointing to a 1-byte binary field containing the process level.

POOLNAME Name of the subpool to be searched and deleted, up to eight bytes in length. Specified as a character string in quotes which is the name of the subpool, or as an address expression pointing to an 8-byte field containing the name of the subpool. Blank names are not permitted. Trailing blanks are insignificant. Default is '00000000'.

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.

Stack On Input

```
            |                       |   Lower
            |                       |   Address
            |0      1    2    3_____|
   0(SP)    |       |               |
            | (1)   | (2) Block Size|
            |_______|_______________|
   4(SP)    |       |               |
            | (3)   | (4) Address of|
            |       | Start of Block|
   8(SP)    |_______________________|
            | (5) Pool name         | 8 bytes
            |_______________________|
  12(SP)    |                       |
            |                       |   Higher
            |_______________________|    Address
  16(SP)    |       Preceding       |
            |      Stack Data       |
```

(1) Option flag:
 Bit 0 0 = Search only at the link level specified in the LINKLEV operand of the FREEHEAP macro.
 1 = Search backward for the subpool name specified in the POOLNAME operand of the FREEHEAP macro, starting from the link level specified in the LINKLEV operand and going backwards until the subpool is found or all the link levels are exhausted.

Bit 1   1 = Delete an entire subpool with the name specified in bytes 8 - 15 of stack and at the link level specified in byte 4 of this input parameter list. A backward search is never initiated if the DELETE flag is set. The SEARCH flag, if specified, is ignored.

Bits 2-7 Reserved; must be zero.

(2) Block size - any length is allowed but the size is rounded up to the nearest 8-byte multiple.

(3) LINK level from which to start searching for the subpool.

0 = Current link level
1 = Parent of the current link level
2 = grandparent of the current link level, etc.
255 = Lowermost link level

(4) Starting address of the block to be deleted.

(5) An 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 buffer location parameters have no meaning and are ignored.

Stack On Output

```
          |                      |    Lower
          |                      |    Address
          |______________________|
 0(SP)    |                      |
          |     Return Code      |    Higher
          |______________________|    Address
          |      Preceding       |
          |      Stack Data      |
```

Output

A return code is issued in the word on top of the stack. The return codes for this macro are as follows:

Code	Description
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 is specified as all blanks, or a nonzero value is in reserved fields.
32	Invalid process level requested.

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)
```

4.2.32 FREESHR - Free Shared Resources (SVC 52)

Syntax

[label] FREESHR

Function

Releases all resources acquired by the user through the sharing task, including held files or records as well as extension rights.

To deallocate a block of memory as requested. All block sizes, including zero, are legal but they are rounded to their nearest 8-byte multiple. This SVC calls FREEFBUF (SVC 6) to deallocate blocks which are greater than or equal to one page length in size. An entire subpool can also be deleted in a single FREEHEAP call, through the use of the DELETE flag. On UNLINK, all the subpools belonging to that link level are automatically deleted. FREEBUF may modify the value in control register 2.

Stack On Input

```
                                          Lower
          |                           |   Address
          |0     1     2     3        |
   0(SP)  |______T_____T______________|
          | (1)  | (2) | (3)          |
          |      |     |______________|
   4(SP)  |                           |
          |                           |   Higher
          |___________________________|   Address
          |       Preceding           |
          |       Stack Data          |
```

(1) Flag byte:

Bit 0	1 = HOLD, 0 = RELEASE.
Bit 1	1 = EXTENSION RIGHTS request.
Bit 2	1 = RELEASE ALL and commit DMS/TX transaction.
Bit 3	1 = TIME OUT in use.
Bit 4	1 = Cancel on error.
Bit 5	1 = Produce ACK GETPARM on error
Bit 6	Reserved for internal use, must be 0.
Bit 7	Reserved for internal use, must be 0.

(2) Time out value in seconds from 0 - 255.

(3) Reserved, must be 0.

Stack On Output

```
                |                       |    Lower
                |0     1     2    3     |    Address
                 ______________________
       0(SP)    |                       |
                |  (1) Return Code      |
                |_______________________|
       4(SP)    |               |       |
                |  (2) User ID  |  (3)  |    Higher
                |_______________|_______|    Address
                |       Preceding       |
                |      Stack Data       |
```

(1) Return code
(2) User ID of user holding extension rights
(3) Unused

Return Codes

Code	Description
0	Success.
4	Timeout holding extension rights. Current extension holder user ID follows.
8	Invalid function sequence (will not occur under DMS/TX).
12	No shared files open by user.
16	Shared not active, or no memory, or XMIT failed.
20	Failed to log transaction end to BIJ, FREEALL not done.
24	Invalid function request parameter.
28	Invalid subtransaction nesting.

Example

```
 RITS    FREESHR
+RITS    PUSHA 0,0              FREE SHARED RESOURCES
+        PUSHA 0,0
+        MVI   0(15),X'20'
+        SVC   52 (FREESHR)
```

4.2.33 FREEXRTS - Free Extension Rights (SVC 52)

Syntax

[label] FREEXRTS

Function

A DMS function that releases extension rights that were previously acquired through the issuing of the GETXRTS.

Stack On Input

```
             |                      |   Lower
             |                      |   Address
             |0    1     2     3    |
    0(SP)    |_____|_____|__________|
             | (1) | (2) |   (3)    |
             |     |     |          |
    4(SP)    |______________________|
             |                      |   Higher
             |                      |   Address
             |______________________|
             |      Preceding       |
             |      Stack Data      |
```

(1) Flag byte:

```
      Bit 0   1 = HOLD, 0 = RELEASE.
      Bit 1   1 = EXTENSION RIGHTS request.
      Bit 2   1 = RELEASE ALL and commit DMS/TX transaction.
      Bit 3   1 = TIME OUT in use.
      Bit 4   1 = Cancel on error.
      Bit 5   1 = Produce ACK GETPARM on error.
      Bit 6   Reserved for internal use, must be 0.
      Bit 7   Reserved for internal use, must be 0.
```

(2) Time out value in seconds from 0 - 255.

(3) Reserved, must be 0.

Stack On Output

```
             |                      |   Lower
             |                      |   Address
             |0    1     2     3    |
    0(SP)    |______________________|
             | (1) Return Code      |
             |                      |
    4(SP)    |______________|_______|
             | (2) User ID  | (3)   |   Higher
             |              |       |   Address
             |______________________|
             |      Preceding       |
             |      Stack Data      |
```

(1) Return code
(2) User ID of user holding extension rights
(3) Unused

Return Codes

Code	Description
0	Success.
4	Timeout holding extension rights. Current extension holder user-id follows.
8	Invalid function sequence (will not occur under DMS/TX).
12	No shared files open by user.
16	Shared not active, or no memory, or XMIT failed.
20	Failed to log transaction end to BIJ, FREEALL not done.
24	Invalid function request parameter.
28	Invalid subtransaction nesting.

Example

```
 FREEUM  FREEXRTS
+FREEUM  PUSHA 0,0                FREE EXTENSION RIGHT
+        PUSHA 0,0
+        MVI   0(15),X'40'
+        SVC   52 (FREEXRTS)
```

4.2.34 GETBUF - Get Buffer Space (SVC 5)

Syntax

```
[label] GETBUF  [LENGTH=(register)][,LEVEL={ address   }]
                                           {(register)}
```

Function

To allocate a buffer area on a 2048-byte (page) boundary. Buffer space is allocated from the low-address end of modifiable data area. Maximum buffer size is restricted only by the size of the caller's modifiable data area, less the user stack size, plus two pages reserved by the Command Processor/Initiator/GETPARM SVC. Control register 2, the system stack limit, may be modified.

Restrictions

For use by certain supervisor call routines and Data Management System routines only. The GETHEAP macroinstruction and SVC have all the functionality of GETBUF, plus additional capabilities. Because blocks are automatically released at program termination when the GETHEAP facility is used, present GETBUF users are encouraged to use the GETHEAP facility.

Parameter Definitions

LENGTH — A register specification in parentheses where the register contains the buffer length. Only lengths which are multiples of 2048 are valid. The default buffer length is 2048.

LEVEL — Process level at which to allocate the block of memory. The default is the current process level. This parameter is available for privileged callers only. The value can be specified as an address expression pointing to a 1-byte binary field containing the process level.

Stack On Input

```
                                          Lower
          |                       |       Address
          |_______________________|
  0(SP)   |   |                   |
          |(1)| (2) Buffer        |
          |   |     Length        |
  4(SP)   |---------------------- |
          | (3) Process Level     |       Higher
          |_______________________|       Address
          |      Preceding        |
          |      Stack Data       |
```

(1) Flag byte: X'80' = set process level

(2) Requested buffer length must be a multiple of the page size (2048).

(3) Process level if flag bit is set. Privileged callers only.

Stack On Output

```
                                              Lower
              |                      |        Address
              |______________________|
  0(SP)       |                      |
              | (1) Return Code      |
              |______________________|
  4(SP)       |                      |
              | (2) Buffer Address   |        Higher
              |______________________|        Address
              |      Preceding       |
              |     Stack Data       |
```

(1) Return code.

(2) Buffer starting address -- If the buffer allocation is unsuccessful, the content of this word is undefined.

Output

A return code is issued in the word on top of the stack, as follows: if the return code is equal to 0, then the next word on the stack contains the address of the buffer allocated. If the return code is nonzero, then the next word on the stack is undefined.

Return Codes

Code	Description
0	Buffer is allocated.
4	Buffer cannot be allocated.
8	Requested length is not a multiple of 2K.
12	Process level requested is greater than caller's level.
16	Internal system error.

Examples

```
 METO     GETBUF
+METO     PUSHA 0,0                First parameter word
+         PUSHA 0,2048             Request default buffer length
+         SVC   5  (GETBUF)


          GETBUF LENGTH=(R4)
+         PUSHA 0,0                First parameter word
+         PUSH  0,R4               Requested buffer length
+         SVC   5  (GETBUF)
```

4.2.35 GETHEAP - Allocate Heap Storage (SVC 56)

Syntax

```
[label] GETHEAP SIZE=(register),[LINKLEV=address,]

                  POOLNAME={ address}[,ROOTLEV][,ALIGN]
                        {'string'}

                  [,SEARCH][,CREATE][,LEVEL={ address   }]
                                           {(register)}
```

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. GETHEAP has all the functionality of GETBUF for allocating page-aligned buffers with the following additional features:

- 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.

- Blocks may be put into different subpools. Advantages of subpooling are that clustering of areas allocated from the same subpool 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.

- 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.

GETHEAP also allows for the sharing of subpools between link levels.

When blocks of memory are allocated, all block sizes including zero are legal. If, however, the block size is not a multiple of eight bytes, the size is rounded up to the nearest 8-byte multiple. Maximum size is restricted only by the caller's modifiable data area size less the size used by the system stack. The space is taken from the low address end of the modifiable data area. 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.

Restrictions

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

Parameter Definitions

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 1-byte field containing the link level in binary. Default is 0 (i.e., current link level).

POOLNAME — The 8-byte name of the subpool to be searched/created. Blank names are not permitted. Trailing blanks are insignificant. Default is '00000000'. Specified as a character string in quotes, or as the address of an 8-byte field containing the name of the subpool.

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 — If specified, a 2048-byte aligned block is returned to the caller. The default is no alignment.

SEARCH — If specified, a backward search for the subpool is initiated starting from the LINKLEV specified. The default is 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.

LEVEL — Process level at which to allocate the block of memory. The default is the current process level. Available for privileged callers only. Specified as an address expression pointing to a 1-byte binary field containing the process level.

Stack On Input

```
                                       Lower
              |                     |  Address
              |0   1    2     3     |
      0(SP)   |____|________________|
              |    |                |
              |(1) | (2) Block Size |
              |____|________________|
      4(SP)   |    |                |
              |(3) | (4) Reserved   |
              |____|________________|
      8(SP)   |                     |
              | (5) POOL Name       | 8 bytes
              |_____________________|
     12(SP)   |                     |
              |                     |
              |_____________________|
     16(SP)   |      Preceding      |  Higher
              |     Stack Data      |  Address
```

(1) Flag byte:
    X'80' = Backward search
    X'40' = Create subpool
    X'20' = Align on page boundary
    X'10' = Select process level
(2) Block Size in bytes.
(3) Link Level.
(4) Reserved.
(5) POOLNAME (eight characters in length).

Stack On Output

```
                                       Lower
              |                     |  Address
              |_____________________|
      0(SP)   |                     |
              | (1) Return Code     |
              |_____________________|
      4(SP)   |                     |
              | (2) Block Address   |  Higher
              |_____or Block Size___|  Address
              |      Preceding      |
              |     Stack Data      |
```

(1) Return code.

(2) Block address - block size:
    For return code = 0, beginning address of allocated block of memory.
    For return code = 4, size of the largest available block of memory.
    For return code > 4, contents are ignored.

Output

A return code is issued in the top word of the stack. The return codes for this macro are as follows:

Code Description

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 modifiable data area. 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.

32 Invalid link level requested.

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)
```

4.2.36 GETPARM - Get Parameters (SVC 20)

Syntax

```
[label] GETPARM [I ,]FORM={REQUEST},KEYLIST={(register)},
                [ID ]     {SELECT }         { address  }
                [R  ]     {ACK    }
                [RD ]     {SYSHDR }
                          {OPR    }

                MSG={(register)}[,DEVICE={(register)}][,SHIFT={NO }]
                    { address  }         { address  }        {YES}

                [,{PFKEYS={   (register)    } }]
                          {     address     }
                          {(ENTER, address)}

                  {PFLIST=(PFK1,PFK2,...PFKn)}
```

Function

To solicit information from the user at a workstation or from a procedure body. Information may be obtained from the user in the following three ways as defined by the FORM parameter:

- By requesting the user to key in necessary modifications to a keyword or a series of keywords.

- By requesting the user to select from a list of functions by depressing the corresponding program function key.

- By requesting the user to press the ENTER key to acknowledge a message.

After displaying the data supplied by the KEYLIST parameter on the workstation screen, GETPARM performs the following actions:

1. Waits for a response from the workstation.

2. Validates the response according to the entry specifications contained in the field format control block for each keyword.

3. If the entry is not valid, GETPARM issues a request for respecification of the invalid data.

4. When all supplied values are valid, GETPARM stores the information in the receiving fields of the field format control block and returns control to the issuing program.

The user program may issue four different types of GETPARMS:

- Initial request
- Respecification
- Initial defaulted
- Respecification defaulted

Fields for which values are requested are identified by a two-level name, prname and keyword, specified in a parameter group control list specified by the KEYLIST parameter. The procedure body, if any, in effect 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 is used to generate a message for display. The KEYLIST macroinstruction is used to generate the parameter group control list addressed by the KEYLIST parameter of GETPARM.

The total number of lines used by KEYLIST and MSGLIST for display can not exceed 18. None of the lines can be longer than 79 characters, excluding the end-of-line character.

The GETPARM SVC enables user programs to request and accept run time parameter information, and to display and wait for acknowledgment of run time messages. GETPARM requests the information or response either through direct interaction with the user through the workstation or by accessing supplied data in the procedure which invoked the program. The program that issues the GETPARM SVC need not be aware of the data source; any interactive program that communicates with the user exclusively, using GETPARM requests, can also be successfully run from a procedure.

GETPARM functions are divided into three request types:

- Request for information -- this request should be issued whenever the user is required to enter data for one or more modifiable receiving fields. The expected user response is to modify none, one, or all of the fields and to end the input of data by pressing the ENTER key.

- Request for selection -- this request should be issued when a list of valid choices is to be displayed and the user is expected to identify a choice by pressing one of the program function keys or the ENTER key.

- Request for acknowledgement -- this request should be issued when the user is to take some operator action, or to acknowledge receipt of a message. In this mode, the user is expected to press the ENTER key as a ready signal.

GETPARM creates and displays a screen full of information using the data structure generated by the KEYLIST macroinstruction whose address is supplied on the call to the SVC. GETPARM generates the screen in the following manner:

- Lines 1 through 6 of the workstation screen, the header section, are reserved for system generated headers which assist the user in responding to the GETPARM request and vary according to request type. These lines include the display area for messages generated by the GETPARM SVC.

- Lines 7 through 24 of the workstation screen contain a user message section and the modifiable fields and embedded text section.

  - The message section is used to display the message supplied to the SVC through the MSG operand of the GETPARM macroinstruction. The message number and the issuer id in the fixed format section of the message are displayed on the screen beginning in column 1 of the first line. The message text is placed on the screen beginning with column 2 of line 7. One additional line beginning in column 2 is displayed for each new-line character encountered. The new-line character is not displayed and does not use a character position. The maximum length of text which can be displayed on one line is 79 characters. The message section is completed with a blank line.

  - The GETPARM SVC builds the rest of the screen according to how the KEYLIST data structure contains the information to be displayed. When GETPARM encounters a field format control block for a keyword/receiving field, it performs in the following manner:

    -- 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

    -- The screen line position advances the indicated number of lines from current position. If line advancement takes place the column position is set to 2.

    -- The screen column position advances the indicated number of spaces from the current position.

-- The 8-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 are not displayed and are not validated. Fields flagged as being in error are blinked to attract user attention.

- When GETPARM encounters a field format control block for embedded text, it performs in the following manner:

  -- The screen line position advances the indicated number of lines from current position. If line advancement takes place the column position is set to 2.

  -- The screen column position advances the indicated number of spaces from current position.

  -- 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 are truncated if they exceed 18 lines of information to be displayed.

Default or current information in the receiving fields is displayed as is without regard for entry control information. However, this information is 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 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, GETPARM automatically generates an error message in the header section, blinks the field in error and issues another read. 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 is saved as is, and the issuance of a CONTINUE command returns control to GETPARM with a restored screen. The user can then reenter the program by completing his response to GETPARM.

The program using GETPARM cannot assume that the user's response (or the accessed procedure data) is valid. The displayable parameters presented to GETPARM include a parameter-group receiving section with embedded 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, bit 6 of the request type indicator, is used by GETPARM to differentiate between an initial request and a request for correction of data which the user program did not consider valid. An initial request, request type I, is satisfied using procedure-specified data (located using the prname and keyword) if available, generating a user-interaction only if all such data has been exhausted. Each initial request with a given prname reads one equivalent specification statement in a procedure. A request for respecification, request type R, always generates a user-interaction regardless of whether or not the program is running from a procedure.

If the user-interaction suppressor bit is set, initial requests access procedure-specified data if available, but do not ordinarily generate a user-interaction. However, if the user has selected the NO DEFAULT option when initiating his program or procedure all GETPARM requests, after accessing available procedure data, are displayed for user observation or modification.

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 coded with the assumption that they must be capable of generating a workstation screen that can be displayed.

Parameter Definitions

I
Indicates initial request for specification of information. 'I' is the default.

R
Indicates request for correction of information, selection, or response just received as a result of the previous request.

RD
Generates a workstation interaction even if the default data in the receiving fields of all field format control blocks satisfy lexical requirements for correctness.

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.

KEYLIST The address of a parameter group control list in the format specified in the GETPARM SVC description. This format is produced by the KEYLIST macroinstruction. This parameter is presented in the same ways as the MSG parameter. A parameter group control list is always required, but, if the user desires, may have no field format control blocks.

MSG The address of a message to be displayed in the user program message section of the workstation screen. 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. The MSG parameter is always required, but the message text may be of 0 length.

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 The address of the program function key mask which indicates which PF keys are accepted with this GETPARM. The address is supplied as a single subparameter or as a subparameter preceded by the word ENTER, as shown in the syntax. When this expression is preceded by ENTER, the ENTER key is also accepted. Otherwise, if just the PFKEYS parameter is supplied, the ENTER key is not accepted. The 4-byte program function key mask is constructed as follows: the high-order bit corresponds to PF key 1, the low-order bit corresponds to PF key 32. Bits set on indicate keys which are accepted. If specified as an address, the address must be in the code section.

May also be specified by designating a register in parentheses where the register contains the PF key mask.

If the PFKEYS parameter is not supplied, the following keys are accepted:

- FORM=REQUEST -- ENTER key only
- FORM=SELECT -- All PF keys and ENTER key
- FORM=ACK -- ENTER key only
- FORM=SYSHDR -- ENTER key only
- FORM=OPR -- ENTER key and PF16 only

PFLIST A list of PF keys which are to be accepted with this GETPARM. PF keys are specified as digits from 1 - 32 separated by commas and enclosed in parenthesis.

SHIFT YES specifies that uppercase PF key numbers should be converted to lowercase PF key numbers. NO specifies no case change and is the default.

Stack On Input

Parameter list of eight or twelve bytes on stack top, in the following format:

```
         |                     |    Lower
         |                     |    Address
         |0     1    2    3    |
 0(SP)   |______T______________|
         | (1)  | (2) Message  |
         |      |     Address  |
 4(SP)   |______T______________|
         | (3)  | (4) Address of|
         |      | Control List |
 8(SP)   |_____________________|
         | (5) PF Key Mask     |    Higher
         |    (Optional)       |    Address
         |_____________________|
         |    Preceding        |
         |    Stack Data       |
```

(1) Request type indicator:
Bits 0-3 -- Header type and 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 PF key. By default, all PF keys are disabled.
1 = Request for selection - Acceptable response is selection indicated and signaled by pressing the ENTER key or any enabled PF key. By default, all PF keys are enabled.
2 = Request for acknowledgement - Acceptable response is acknowledgement signaled by pressing the ENTER key or any enabled PF key. By default, all PF keys are disabled.

Bit 4: 1 = Indicates that the ENTER key is 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 -- PF key mask present indicator.

0 = Default mode - PF keys are enabled or disabled according to default values. The PF key mask should not be present in the parameter list.

1 = Override mode - PF keys are disabled as indicated by the PF key 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 generates a workstation interaction even though the default data in the receiving fields of all field format control blocks are correct. No workstation interaction is generated when procedure-specified data is supplied 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 does not generate a workstation interaction unless a field default value is lexically in error.

(2) Address of the message to be displayed. The message is constructed as follows:

(a) Message Number	(b) Issuer ID	(c) Length	(d) Text	
0	4	10	12	N

(a) A 4-byte message number in ASCII characters. This number is displayed in the GETPARM header on any associated screen transactions.

(b) A 6-byte issuer ID in ASCII characters. This ID is displayed in the GETPARM header on any associated screen transactions.

(c) A 2-byte message length in binary. This is the length of the text that follows.

(d) Message text in ASCII characters. If the message is longer than 79 characters, (meaning multiple display lines) an end of line can be indicated by an ASCII new-line character (X'0D'). The message text is displayed on the workstation beginning in column 2 of line 7. Each new line begins in column 2 of the next line. Lines longer than 79 characters containing no new-line character are truncated. The last line does not require an end-of-line indicator.

(3) Zero, or for device action request only, the device number of the device requiring service.

(4) Address of a parameter group control list which is constructed as follows:

```
                 _______________________________
       +0       |                               |
                | (a)  prname                   |
                |_______________________________|
       +8       | (b) PF Key  |  (c) Number     |
                |     Field   |      of fields  |
      +10       |_____________|_________________|
                | (d)  Field Format             |
                |      Control Block  1         |
                |                               |
                |_______________________________|
  +10 + BL      | (d) Field Format              |
          1     |     Control Block  2          |
                |_______________________________|
                |              .                |
                |              .                |
                |              .                |
                |              .                |
                |              .                |
                |              .                |
     n-1        |______________.________________|
+10 +    BL     |                               |
           i    | (d) Field Format              |
     i=1        |     Control Block N           |
                |                               |
                |_______________________________|
```

where BL = length of Format Control Block

(a) An 8-character, left-justified parameter reference name (prname).

(b) A 1-byte receiving field for the corresponding AID character of the program function key received in a user response to a request for selection. This field may be set by a procedure specification of a function key number.

(c) A 1-byte binary count - number of field format control blocks.

(d) Format control block (variable length field). There are two formats for the format control blocks: one for control of the keyword/receiving field pairs, the other to control the use of embedded text to be displayed. This field is repeated for each field to be displayed in the order they are to be displayed on the workstation screen.

(5) Optional 4-byte program function key mask. Each bit indicates whether the corresponding PF key should be enabled. The high-order bit corresponds to PF1. A value of 1 = PF key enabled; 0 = PF key disabled.

Keyword/Receiving Field Format Control Block Structure

```
     Data Structure
 ____________________
|0    1    2    3    |
|    |    |    |     |      Lower
| (1) | (2)| (3)| (4)|      Address
|_____|____|____|____|
|                    |
| (5)  Keyword       |
|____________________|
|                    |
| (6)  Receiving     |
|      Field         |      Higher
|____________________|      Address
```

(1) Line-advance-count for display control. A 1-byte binary field.

(2) Space-advance-count for display control. A 1-byte binary field. Line advance takes place before space advance. Both take place before display of keyword and receiving field.

(3) Field error flag and receiving field entry restriction indicator. A 1-byte binary field, formatted as follows:

Bit 0: Field error flag

1 = Error - set by program to draw attention to fields in error. Reset by GETPARM.

Bit 1: 1 = Nonmodifiable field.

Bits 5-7: Receiving field 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 are 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 are ignored. An all blank response is treated as a legitimate NULL response. Field length is restricted to 16 characters.
4 = Uppercase alphanumeric - all entered letters are converted to uppercase. Legal nonblank response must be left-justified and consist entirely of the numerals 0-9, the letters A-Z, the special characters (@, #, or $), and trailing blanks. An all blank response is treated as a legal NULL response indicator. Maximum usable field length is 68 characters.
5 = Uppercase hexadecimal - all entered letters are converted to uppercase. 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. An all blank response is 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 are converted to uppercase. Legal nonblank response is left-justified, beginning with a letter A-Z, or one of the special characters (@, #, or $), and consists entirely of the numerals 0 through 9, the letters A through Z, the special characters, and trailing blanks. An all blank response is treated as a legal NULL response indicator. Maximum usable field length is 68 characters.

(4) 1-byte binary receiving field length minus 1 (in characters).

(5) An 8-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.

Embedded Text Field Format Control Block Structure

```
      DATA STRUCTURE
 ______________________
|0   1    2    3       |
|____|____|____|_______|
|    |    |    |       |      Lower
| (1)| (2)| (3)|  (4)  |      Address
|____|____|____|_______|
|                      |
| (5) Text             |
|                      |
|______________________|      Higher
                              Address
```

(1) Line-advance-count for display control. A 1-byte binary field.

(2) Space-advance-count for display control. A 1-byte binary field. Line advance takes place before space advance. Both take place before display of keyword and receiving field.

(3) The value -1 (= 255).

(4) Text field character length minus one. A 1-byte binary field.

(5) Character string to be displayed. Variable length field.

Stack On Output

The stack on output is empty, except for the top word which contains a return code. When the return code equals 0, the operation was successful. If there was an error, the return code equals 04.

```
                                              Lower
                                              Address
            |                      |
            |______________________|
  0(SP)     |                      |
            |                      |
            |______________________|
  4(SP)     |                      |
            |                      |      Higher
            |______________________|      Address
            |       Preceding      |
            |      Stack Data      |
```

(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 0.

(4) Input parameters popped from stack upon return.

Example

```
LAB1     GETPARM KEYLIST=(R2),MSG=LAB2
LAB1     PUSH  0,R2                    Put the KEYLIST address on the stack
         PUSHA 0,LAB2                  Put the MSG address on the stack
         MVI   0(15),B'00000000'       Move in the GETPARM options byte
         SVC   20 (GETPARM)            Issue the GETPARM SVC

LAB2     MSGLIST  '1234','TXTEDT','OPTIONS AS FOLLOWS:'
LAB2     DC    CL4'1234' MESSAGE NUMBER
         DC    CL6'TXTEDT' ISSUER IDENTIFICATION
         DC    AL2(19)   MESSAGE LENGTH
         DC    C'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)
LAB3     DC    CL8'OPT'   PRNAME
         DC    HL1'0,1'                PF KEY   &   FIELD COUNT
          DC   AL1(1,0,4,1)                 LINE / SPACE ADV, FLAGS, LGTH-1
         DC    CL8'LIST',CL2'NO'                  KEYWORD & DISPLAYED VALUE
         END BEGIN
```

4.2.37 GETXRTS - Hold Extension Rights (SVC 52)

Syntax

```
[label] GETXRTS TIMEOUT={(register)}
                        { integer  }
```

Function

A DMS function that provides a way for a user to acquire more resources when already holding some resources. Only one user at a time may have extension rights. This avoids the possibility of deadlock occurring when more than one user requests the same resource at the same time.

Parameter Definitions

TIMEOUT — Specifies the time (in seconds) to wait for extension rights. An integer value from 0 to 255 seconds, where 0 equals no limit on the time to wait. The value may also be specified as a register number in parenthesis which contains the wait value.

Stack On Input

```
             |                      |   Lower
             |                      |   Address
             |0    1     2    3_____|
   0(SP)     |_____T_____T__________|
             | (1) | (2) |   (3)    |
             |_____|_____|__________|
   4(SP)     |                      |
             |                      |   Higher
             |______________________|   Address
             |      Preceding       |
             |      Stack Data      |
```

(1) Flag byte:

Bit 0	1 = Hold; 0 = Release.
Bit 1	1 = Extension rights request.
Bit 2	1 = Release all and commit DMS/TX transaction.
Bit 3	1 = Time out in use.
Bit 4	1 = Cancel on error.
Bit 5	1 = Produce ACK GETPARM on error.
Bit 6	Reserved for internal use, must be 0.
Bit 7	Reserved for internal use, must be 0.

(2) Time out value in seconds from 0 - 255.

(3) Reserved, must be 0.

Stack On Output

```
               |                        |    Lower
               |                        |    Address
               |0     1     2     3     |
      0(SP)    |________________________|
               |                        |
               | (1) Return Code        |
               |________________________|
      4(SP)    |               |        |
               | (2) User ID   |  (3)   |    Higher
               |_______________|________|    Address
               |      Preceding         |
               |     Stack Data         |
```

(1) Return code
(2) User ID of user holding extension rights
(3) Unused

Return Codes

Code	Description
0	Success.
4	Timeout holding extension rights. Current extension holder user ID follows.
8	Invalid function sequence (will not occur under DMS/TX).
12	No shared files open by user.
16	Shared not active, or no memory, or XMIT failed.
20	Failed to log transaction end to BIJ, FREEALL not done.
24	Invalid function request parameter.
28	Invalid subtransaction nesting.

Example

```
 GETUM    GETXRTS  TIMEOUT=3
+GETUM    PUSHA 0,0                  HOLD EXTENSION RIGHT
+         PUSHA 0,0
+         MVI   0(15),X'D0'          TIME-OUT IN USE
+         MVI   1(15),3              SET UP TIME-OUT INTERVAL
+         SVC   52 (HOLD)
```

4.2.38 HALTIO - Halt I/O Operation (SVC 12)

Syntax

Format 1:

```
[label] HALTIO  PRINTER={(register)}
                        {  integer }
                        {  address }
```

Format 2:

```
[label] HALTIO  OFB={(register)}
                    { address   }
```

Format 3:

```
[label] HALTIO  VCB={(register)}
                    { address   }
```

Function

Stops an input/output operation that was initiated by the XIO SVC. The printer option terminates multiline (especially block-oriented) print I/O requests to a printer. The OFB option terminates an outstanding file-oriented I/O request which is not necessarily for a printer output file (especially for telecommunications files). The VCB option terminates an outstanding volume-oriented I/O request to or from a disk.

If the I/O operation is in progress and the device supports HALT I/O, the SVC issues an HIO instruction to the device. If the I/O is not in progress, the IORE is removed from the device's I/O queue.

In either case, a WAIT must be issued after the call to HALTIO, to wait for the I/O completion, clean up the device status, and leave the completion semaphore at the correct value.

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).

Restrictions

HALTIO is intended for system routine use and those user programs which must control I/O operations through XIO (Execute Physical I/O). HALTIO is 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 insure the HALTIO is complete by issuing a subsequent CHECK I/O macroinstruction.

Parameter Definitions

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 that contains the device number in binary in its low-order position, as an integer which is the device number in decimal, or as an expression that addresses a 1-byte binary field containing the device number.

OFB The address of the open file block (OFB) for the outstanding I/O. This form is used for file-oriented (regular) I/O and may reference any file/device pairing. This parameter is specified as a register in parentheses containing the OFB address in the low-order three bytes, or as an address expression pointing to a 4-byte field containing the OFB address in the low-order three bytes.

VCB The address of the volume control block (VCB) for the outstanding I/O. This form is used for volume-oriented (VOLIO) I/O on disk devices only. This parameter is specified as a register in parentheses containing the VCB address in the low-order three bytes, or as an address expression pointing to a 4-byte field containing the VCB address in the low-order three bytes. The macroinstruction code appends +1 to the specified VCB address in order to differentiate (for HALTIO SVC processing) an OFB address from a VCB address.

Output

For the PRINTER option, the HALTIO SVC (SVC 12) issues a return code in the top word of the stack. This return code corresponds to the condition code set by the HIO machine instruction (refer to VS Principles of Operation).

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

Stack On Input

Device Option

```
                                            Lower
               |                      |     Address
               |0     1     2     3___|
       (SP) |  |______________________|
               |      |          |    |
               | (1)  |          |(2) |     Higher
               |______|__________|____|     Address
               |      Preceding       |
               |      Stack Data      |
```

(1) Byte 0 = 0
(2) Byte 3 = Device number

OFB and VCB Option

```
                                               Lower
            |                       |          Address
            |0     1     2    3     |
     (SP) | |      |________________|
            | (1)  | (2)            |          Higher
            |______|________________|          Address
            |      Preceding        |
            |     Stack Data        |
```

(1) Byte 0 = X'80'
(2) OFB option: Bytes 1-3 = OFB address
    VCB option: Bytes 1-3 = VCB address + 1

Stack On Output

```
                                               Lower
            |                       |          Address
            |_______________________|
     (SP) | |                       |
            | (1) Condition Code    |          Higher
            |_______________________|          Address
            |      Preceding        |
            |     Stack Data        |
```

Device Option

(1) Condition code returned from HIO instruction.

OFB or VCB Option

The SVC produces no output and pops the input parameters off the stack.

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

 LAB      HALTIO VCB=VCBADDR
+LAB      PUSHC    0(4),VCBADDR       PUSH VCB ADDRESS ONTO STACK
+         OI       3(15),X'01'        FLAG AS VCB
+         MVI      0(15),X'80'        FLAG AS OFB/VCB TYPE PARMLIST
+         SVC      12  (HALTIO)       ISSUE SVC
```

4.2.39 IPCB - Describe Interprocessor Control Block

Syntax

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

Function

Describes the interprocessor control block (IPCB), a variable length block. Its length is 4 + (8*N), where N = number of devices. These devices are used to communicate with a peripheral processor (DLP).

Parameter Definitions

NODSECT — Specification of NODSECT results in the IPCB fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, the system generates a DSECT with the name IPCB (plus optional suffix).

REG — Provides for the optional specification of a register for which a USING statement for the IPCB fields is generated.

SUFFIX — One ASCII character in length. If provided, all labels are generated by the concatenation of the letters IPCB, the user-provided SUFFIX and the field name.

Structure

IPCB

		BYTE 0	BYTE 1	BYTE 2	BYTE 3
	+0			DCNT	
SLOT |	+4	SPARE		DEV	
	+8	OFB			
	+C				

Example

```
    IPCB REG=3,SUFFIX=E
+IPCBE         DSECT
+*
+*  THE INTER PROCESSOR CONTROL BLOCK (IPCB) IS A VARIABLE LENGTH
+*  BLOCK; ITS LENGTH IS 4+(8*N), WHERE N = NUMBER OF DEVICES.
+*  THESE DEVICES ARE USED (BY TASKS IN THE VS) TO
+*  COMMUNICATE WITH A PERIPHERAL PROCESSOR (OR 'DLP').
+*
+*  DATE: JULY 16, 1980
+*
+IPCBEBEGIN        DS  OF                  (ALIGNMENT)
+                  DS  H                   (RESERVED; MUST BE ZERO)
+IPCBEDCNT         DS  H                   NUMBER OF DEVICES
+IPCBESLOT         DS  XL8
+                  ORG IPCBESLOT
+                  DS  H                   SPARE
+IPCBEDEV          DS  H                   VS DEVICE ADDRESS
+IPCBEOFB          DS  A                   OFB ADDRESS
+IPCBEEND          EQU *                   END OF IPCB
+IPCBELENGTH       EQU IPCBEEND-IPCBEBEGIN
+BEGIN             CODE
+                  USING IPCBE,3
```

4.2.40 IPCLOSE - Close For I/O with Telecommunications Devices or Data Link Processor (SVC 50)

Syntax

```
[label] IPCLOSE IPCB={ address   }
                     {(register)}

                [,DEVICEADDR={     address      }]
                             {    (register)    }
                             {self-defining term}

                [,NUMBER=    {     address      }]
                             {    (register)    }
                             {self-defining term}

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

Function

The IPCLOSE macroinstruction would normally be used by communication control programs or emulator support programs to close the I/O facility (i.e., the channel devices) between the VS and the Data Link Processor (DLP). The conceptual channels between the VS and the DLP that have been opened by the IPOPEN macroinstruction must be closed when I/O processing has been completed by the task. The channel devices are closed by using the IPCLOSE macroinstruction. If RELEASE=YES is specified, IPCLOSE is also used to release any exclusive reservation of the devices that were exclusively reserved through IPOPEN.

IPCLOSE closes the specified number of devices in the specified IPCB, starting with the specified device address. Thus, the caller can close one or a number of channel devices. The caller must have opened the devices specified (using IPOPEN), the IPCB must contain entries in the proper format, and the devices specified must correspond to the device address/OFB information in the IPCB. The information contained in the IPCB is sufficient for the IPCLOSE SVC to close the devices previously opened, therefore only the IPCB address need be specified by the user; the DEVICEADDR and NUMBER parameters are optional.

A HALTIO is issued for any outstanding interprocessor I/O operations.

All OFBs and IOREs for the specified devices are unlinked and freed. The entries in the IPCB corresponding to the closed devices are zeroed out. If the last active device on the DLP is being closed, the DLP is marked as not loaded.

When RELEASE=YES, the device(s) are released at IPCLOSE time. If RELEASE=YES is specified, and any DLP for any of the devices has been reserved by the user through UNITRES, then IPCLOSE does not complete the operation. The UNITRES macroinstruction must be used to release the devices of a DLP previously reserved by UNITRES.

No device is released when RELEASE is not specified as YES. Therefore, a task retains exclusive reservation of the devices reserved by IPOPEN even if the devices are not open any longer for I/O processing. The user can subsequently open these devices with RESERVE=NO (the default option) specified in the IPOPEN macroinstruction.

IPCLOSE does not CANCEL under any circumstances.

Parameter Definitions

IPCB — A required parameter that defines the address of an interprocessor control block (IPCB). The IPCB address need not be the same as used in the corresponding IPOPEN macro that initiated the I/O facility; this means that the user, after IPOPEN, can move the IPCB to any other area in the user modifiable area, and present this new address to IPCLOSE. Also, many IPCBs can be combined in one IPCB, and all devices in the combined IPCBs closed in one IPCLOSE call; therefore, devices on different DLPs can be closed at one time.

The value can be specified as an address expression, or as a register in parentheses containing the address of the IPCB in the low-order three bytes. This parameter is required.

DEVICEADDR — An optional parameter that defines the starting device address. It can be specified as a self-defining term, as a register in parentheses that contains the device address in its low-order two bytes, or as an expression that addresses a 2-byte field containing the device address. If not specified, the starting device address is taken as the first device in the specified IPCB.

NUMBER — An optional parameter that specifies the number of device addresses to be closed. If NUMBER is not specified, then the IPCLOSE uses the number of devices indicated in the specified IPCB. Specified as a self-defining term, as a register in parentheses containing the number in its low-order two bytes, or as an expression addressing a 2-byte field containing this number.

RELEASE — When RELEASE=YES the devices are released as they are closed; the default is NO.

Stack On Input

```
                |                       |   Lower
                |0    1     2    3      |   Address
       0(SP)    |     |     |           |
                | (1) | (2) | (3)       |
                |     |     |           |
       4(SP)    |           |           |
                | (4)       | (5)       |
                |           |           |
       8(SP)    |                       |
                | (6) IPCB Address      |
                |                       |
      12(SP)    |                       |
                | (7) DLP Name          |
                |                       |
      16(SP)    |                       |
                | (8) Spare             |   Higher
                |                       |   Address
                |       Preceding       |
                |      Stack Data       |
```

(1) Function code (1 byte)
    X'00' - Open
    X'01' - Close

(2) Option code (1 byte)
    X'00' - No RES/REL
    X'01' - RES/REL

(3) Number of device addresses (2 bytes)

(4) Reserved, must be zero (2 bytes)

(5) First device address or zero (2 bytes)

(6) IPCB address (4 bytes)

(7) DLP name (unused for CLOSE function) (4 bytes)

(8) Spare (4 bytes)

Stack On Output

```
                                                Lower
             |                          |       Address
             |0    1     2     3  ______|
   0(SP)     |     |                    |
             | (1) | (2) Device         |
             |_____|     Address________|
   4(SP)     |                          |
             |                          |       Higher
             |__________________________|       Address
             |       Preceding          |
             |      Stack Data          |
```

(1) Return code
(2) Device address

Output

A return code is returned to the caller to indicate the overall success or failure of the IPCLOSE processing. Any error causes nothing to have occurred, that is, if successful, the operation is completely successful. If the IPCLOSE processing is not successful, the first device in error which caused the operation to be terminated without completion is indicated in the low-order two bytes of the word returned to the caller on the stack. The specific error condition is indicated by the value of the high-order byte of the word returned to the caller on the stack.

Return Codes

Code	Description
0	IPCLOSE successful.
4	Invalid device address for starting address.
8	Invalid IPCB (address or contents).
12	No devices OPEN or IPOPEN.
16	Device not OPEN or IPOPEN.
20	Invalid release option (IPCLOSE attempted with RELEASE=YES and DLP reserved by the user; this return code is intended to alert the user to release the DLP by a call of the UNITRES SVC).

Example

```
          IPCLOSE IPCB=IPCBLK,DEVICEADDR=(R2),NUMBER=(R3),RELEASE=YES
+         PUSHN    0,8                GET EIGHT BYTES ON THE STACK
+         XC       0(8,15),0(15)      AND ZERO OUT SPARE BYTRS
+         PUSHA    0,IPCBLK           PUSH ADDRESS OF THE "IPCB"
+         PUSHA    0,0                CLEAR 4 BYTES OF STACK SPACE
+         STH      R2,2(,15)          SET FIRST DEVICE ADDRESS
+         PUSHA    0,0                CLEAR 4 BYTES OF STACK SPACE
+         STH      R3,2(,15)          SET NUMBER OF DEVICES TO CLOSE
+         MVI      1(15),1            SET RELEASE = YES OPTION
+         MVI      0(15),1            INDICATE "IPCLOSE"
+         SVC      50                 (IPOPEN/IPCLOSE)
```

4.2.41 IPOPEN - Open for I/O with Telecommunications Devices or Data Link Processor (SVC 50)

Syntax

```
[label] IPOPEN  IPCB={ address  },DEVICEADDR={     address      },
                     {(register)}             {    (register)    }
                                              {self-defining term}

                NUMBER={     address      },
                       {    (register)    }
                       {self-defining term}

                DLPNAME={     address      }
                       {    (register)    }
                       {'character string'}

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

Function

IPOPEN is used by communication control programs or emulator support programs to open the I/O facility (i.e., channel devices) between the VS and the data link processor (DLP). The conceptual channels between a program in the VS and a DLP must be opened before any I/O is attempted.

Prior to calling the IPOPEN SVC, the programmer is responsible for allocating an area of the program's modifiable data area for the interprocessor control block (IPCB). The user places the address of previously-allocated IPCB in the IPCB parameter.

The caller designates what devices to open by specifying the starting device address in the DEVICEADDR parameter, by specifying the number of devices to be opened in the NUMBER parameter, and by specifying the 4-byte DLP name (assigned at system generation) in the DLPNAME parameter. The channel device addresses associated with a DLP are extracted using the EXTRACT macroinstruction.

If IPOPEN is successful, it places the count of devices, device addresses, and corresponding OFB addresses into the IPCB whose address is specified in the IPCB parameter. If the RESERVE=YES option was specified, IPOPEN also reserves the opened devices.

IPOPEN cancels if the IPCB address is invalid. The cancellation condition and the corresponding message are "50#0: Invalid IPCB".

If IPOPEN fails for any reason, IPOPEN closes all devices that had been opened up to the point the error condition was detected.

Parameter Definitions

IPCB — A required parameter that defines the address of the area for the interprocessor control block (IPCB) which must be allocated by the user prior to issuing the IPOPEN macroinstruction. The value can be specified as an address expression, or as a register in parentheses that contains the address of the IPCB in the low-order three bytes.

DEVICEADDR — An optional parameter that defines the starting device address. The value can be specified as a self-defining term which is the device address, as a register in parentheses that contains the device address in its low-order two bytes, or as an expression that addresses a 2-byte field containing the device address. If not specified, the starting device address is taken to be the first device configured on the DLP specified by the DLPNAME parameter.

NUMBER — A required parameter that specifies the number of device addresses to be opened. The value can be specified as a self-defining term which is the number of devices, as a register in parentheses that contains the number in its low-order two bytes, or as an expression that addresses a 2-byte field containing this number.

DLPNAME — A required parameter that specifies the name of the DLP. The value can be specified as a 4-byte character string in single quotes, as a register in parentheses that contains the name, or an expression that addresses a 4-byte field containing this name.

The DLPNAME is associated with the DLP at SYSGEN.

RESERVE — When RESERVE=YES, the devices are exclusively reserved as they are opened. The default is no reservation.

Stack On Input

```
               |                     |   Lower
               |0    1     2    3    |   Address
  0(SP)        |     |     |         |
               | (1) | (2) | (3)     |
               |     |     |         |
  4(SP)        |           |         |
               | (4)       | (5)     |
               |           |         |
  8(SP)        |                     |
               | (6) IPCB Address    |
               |                     |
 12(SP)        |                     |
               | (7) DLP Name        |
               |                     |
 16(SP)        |                     |
               | (8) Spare           |   Higher
               |                     |   Address
               |      Preceding      |
               |     Stack Data      |
```

(1) Function code (1 byte)
  X'00' - Open
  X'01' - Close

(2) Option code (1 byte)
  X'00' - No RES/REL
  X'01' - RES/REL

(3) Number of device addresses (2 bytes)

(4) Reserved, must be 0 (2 bytes)

(5) First device address or 0 (2 bytes)

(6) IPCB address (4 bytes)

(7) DLP name (unused for CLOSE function) (4 bytes)

(8) Spare (4 bytes)

Stack On Output

```
                                            Lower
            |                      |        Address
            |0    1     2     3    |
  0(SP)     |     |                |
            | (1) |                |
            |     |                |
  4(SP)     |                      |
            |                      |        Higher
            |                      |        Address
            |      Preceding       |
            |      Stack Data      |
```

(1) Return code

Return Codes

Code	Description
0	IPOPEN successful.
4	Invalid device address (starting address).
8	Specified number of devices is unavailable on the specified DLP.
12	Invalid DLP name.
16	The DLP is reserved by another task.
20	Insufficient system memory pool (GETMEM failure) for OFB or IORE allocation.
24	Maximum number of OFBs exceeded.
28	Invalid reserve option (IPOPEN attempted when RESERVE=YES and DLP already reserved by the user).

Example

```
         IPOPEN  IPCB=IPCBLK,DEVICEADDR=(R2),DLPNAME=(R4),RESERVE=YES,  -
                 NUMBER=IPNUM
+        PUSHA   0,0                 ZERO OUT SPARE BYTES
+        PUSHA   0,0                 CLEAR 4 BYTES OF STACK SPACE
+        MVC     0(4,15),0(R4)       SET DLPNAME
+        PUSHA   0,IPCBLK            PUSH ADDRESS OF THE "IPCB"
+        PUSHA   0,0                 CLEAR 4 BYTES OF STACK SPACE
+        STH     R2,2(,15)           SET FIRST DEVICE ADDRESS
+        PUSHA   0,0                 CLEAR 4 BYTES OF STACK SPACE
+        MVC     2(2,15),IPNUM       SET NUMBER OF DEVICES TO OPEN
+        MVI     1(15),1             SET RESERVE = YES OPTION
+        SVC     50                  (IPOPEN/IPCLOSE)
```

4.2.42 KEYLIST - Generate Parameter Group Control List

Syntax

```
[label] KEYLIST PRNAME='name',[LABELPFX='prefix'][,PREVIEW={YES}]
                                                   {NO }

        {'keyword',({'default-value'}[,{CHAR }][,line-advance]
                    {absolute-length}  { INT }
                                       { NUM }
                                       { AN  }
                                       { HEX }
                                       {UCHAR}
                                       { ANL }

                   [,space-advance]),                               }

        {TEXT,('display-text'[,line-advance][,space-advance]),    }
        {textname,('display-text'[,line-advance][,space-advance]),}
```

Function

Generates a data structure suitable for use with the KEYLIST parameter of the GETPARM macroinstruction. The data structure, called a parameter group control list, is used by the GETPARM SVC to display a screen full of information on the workstation and to save the user response. The keyword, TEXT, and textname formats may be repeated as often as necessary to define the contents of the screen.

Parameter Definitions

PRNAME — Parameter reference name, a name that identifies the parameter group which is associated with one screen of information. It can be up to eight characters in length; characters can be alphanumeric; the first character must be alphabetic .

LABELPFX — A character string in quotes that prefixes each keyword name and the resulting string used to label each corresponding field format control block. The label is placed on the line-advance byte. Thus, for the keyword/receiving field format control block, the flag byte is at the location specified by this label +2, and the receiving field ('displayed-value') is at this location +12. This parameter is optional.

PREVIEW — If YES is specified, a simulated screen display is printed in the source listing (via comment level MNOTES), in the format specified by the macroinstruction parameters. If NO is specified, the display is not generated in the listing, and "CURRENT LINE LENGTH" messages are not generated. NO is the default.

keyword — A name of one to eight alphanumeric characters enclosed in single quotes which identifies a specific parameter within the group. Specification of keyword is mutually exclusive with specification of TEXT or text name.

default-value — A character string in single quotes containing the default value for the keyword parameter which is displayed with the keyword on the workstation screen. 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 default-value is mutually exclusive with specification of absolute-length.

absolute-length — An absolute expression defining the length of the receiving field for this parameter. Specification of absolute-length is mutually exclusive with specification of default-value.

NOTE

Leading and trailing blanks are accepted in any of the following formats except alphanumeric (AN) and limited alphanumeric (ANL), where only trailing blanks are accepted.

CHAR — Any character is accepted as a valid response.

INT — Only unsigned integers are accepted as a valid response.

NUM — Numbers, with optional decimal point or leading sign, or both, are accepted as a valid response.

AN — Letters, including the special characters #, @, and $, and numerals are accepted as valid response. GETPARM converts any lowercase letters to uppercase.

HEX — Only numerals and letters A-F are accepted as a valid response. Lowercase letters a-f are converted to uppercase.

UCHAR — Any character is accepted as a valid response. Lowercase letters are converted to uppercase.

ANL — Letters, including the special characters #, @, and $; and numerals are accepted as a valid response. GETPARM converts lowercase letters to uppercase. The first character cannot be a number.

line-advance The number of lines to advance before displaying the keyword and receiving field, or the embedded text. Valid range is from 0 to 18 and if not specified, defaults to 1. If nonzero, the keyword or text is displayed starting on the current line plus the line advance value. If 0, line advancing does not occur.

space-advance The number of spaces to advance within a line before displaying the keyword. Valid ranges is from 0 to 78. The default is 0. If specified or omitted, the value of space-advance plus 1 is the number of spaces that appear on the workstation screen between either the previous field (if 0 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', where "nn" represents one or two digits with a value no less than 2 and no greater than 80 that indicates the absolute column in which the field is to begin. The appropriate field-advance value is calculated and placed in the field format control block.
- 'CENTER', where the appropriate field-advance value is calculated and placed in the field format control block such that the field is centered within the 80-column workstation screen line.
- 'RIGHT', where the appropriate field-advance value is calculated and placed in the field format control block such that the field is right-justified on the 80-column workstation screen line.

Regardless of how the space-advance is specified, an 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.

TEXT Indicates that embedded text is supplied in the next parameter.

textname A nonquoted text /ame 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. Specification of TEXT or textname is mutually exclusive with specification of 'keyword'.

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

Example

```
 PR1          KEYLIST   PRNAME='OPT',LABELPFX='A',PREVIEW=YES,             -
                   'LIST',('NO ',AN),                                      -
                   'DISPLAY',('YES',AN,0,5),                               -
                   TEXT,('NUMBER OF LINES'),                               -
                   'LINECNT',('50',INT,0,5),                               -
                   KOPIES,('NUMBER OF COPIES',0,5),                        -
                   'COPIES',('1',INT,0,5)
+PR1DC        CL8'OPT'                      PRNAME
+             DC   HL1'0,6'                 PF KEY   &   FIELD COUNT
+ALIST        DC   AL1(1,0,4,2)             LINE / SPACE ADV, FLAGS, LGTH-1
+             DC   CL8'LIST',CL3'NO '       KEYWORD & DISPLAYED VALUE
+                  **  CURRENT LINE LENGTH IS 15  **
+ADISPLAY     DC   AL1(0,5,4,2)             LINE / SPACE ADV, FLAGS, LGTH-1
+             DC   CL8'DISPLAY',CL3'YES'    KEYWORD & DISPLAYED VALUE
+                  **  CURRENT LINE LENGTH IS 35  **
+             DC   AL1(1,0,-1,14)           LINE / SPACE ADV, FLAGS, LGTH-1
+             DC   C'NUMBER OF LINES'       DISPLAYED TEXT
+                  **  CURRENT LINE LENGTH IS 16  **
+ALINECNT     DC   AL1(0,5,1,1)             LINE / SPACE ADV, FLAGS, LGTH-1
+             DC   CL8'LINECNT',CL2'50'     KEYWORD & DISPLAYED VALUE
+                  **  CURRENT LINE LENGTH IS 35  **
+             DC   AL1(0,5,-1,15)           LINE / SPACE ADV, FLAGS, LGTH-1
+KOPIES       DC   C'NUMBER OF COPIES'      DISPLAYED TEXT
+                **  CURRENT LINE LENGTH IS 57  **
+ACOPIES      DC   AL1(0,5,1,0)             LINE / SPACE ADV, FLAGS, LGTH-1
+             DC    CL8'COPIES',CL1'1'       KEYWORD & DISPLAYED VALUE
+                **  CURRENT LINE LENGTH IS 75  **
+   PREVIEW OF KEYLIST "PR1"
+************************************************************************
+         1         2         3         4         5         6         7
+123456789012345678901234567890123456789012345678901234567890123456789012
+************************************************************************
+
+
+ LIST      = NO      DISPLAY  = YES
+ NUMBER OF LINES      LINECNT  = 50      NUMBER OF COPIES     COPIES  = 1
+
+************************************************************************
+         1         2         3         4         5         6         7
+123456789012345678901234567890123456789012345678901234567890123456789012
+************************************************************************
```

4.2.43 LINK - Link to Another Program (SVC 4)

Syntax

```
[label] LINK [{EP='string'  }][,SYSTEM][,NOFAIL][,LOADONLY][,XPARENT={YES}]
              {EPLOC=address}                                        {NO }

             [,LIBRARY={address }][,VOLUME={address }][,RESTRICT={YES}]
                       {'string'}          {'string'}           {NO }

             [,PERM={YES}][,NOPERM={YES}]
                    { NO}          { NO}
```

Function

The LINK macroinstruction initiates the execution of another program from within a currently active program. For example, the operating system's command processor uses LINK to initiate execution of requested programs and the system's procedure interpreter.

The service performs the following functions:

- Parameters to the LINK SVC are pushed onto the program stack and the LINK SVC code is executed.

- If linking to a program, LINK searches the user's default library and volume (which have been specified by the SET or RUN command) for the executable program file for the program specified by the EP parameter. If the SYSTEM parameter is specified, LINK only searches the system disk for the file. An alternate library and volume may be specified through the LIBRARY and VOLUME parameters on the LINK macro. The invoked program returns to the invoker by means of the RETURN macroinstruction. Execution of the LINK macroinstruction pushes status information onto the stack, as well as creating the static areas for the linked-to 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.

- A new program exception element is built for the new program indicating that no program exception exits are established (set). Any user program exception exit previously set by the PCEXIT SVC is restored when an UNLINK is issued to return to the linked-from program.

- Static storage areas for the new program are allocated and register 14 points to the beginning of this area on entry to the linked-to program. Static areas, as defined, are static only within a group of program modules that are linked together, and are removed from the stack by execution of the UNLINK supervisor call routine.

- If linking to a subroutine, register 1, by convention, addresses a standard argument list to be passed to the routine invoked as a result of the LINK and is preserved across the link. The argument list must not be in the reentrant code area.

- For control purposes, the LINK SVC writes the address of the UNLINK routine, the return address to be executed on RETURN, on the stack. This is followed by a back link chain word and register save area for use with the RETURN macroinstruction which executes the RTC instruction. Control register 1 contains the address of this save area (as does general register 15) on completion of the LINK.

- Register 14 addresses the first (doubleword-aligned) address of the newly created group of static areas. Other register contents (except register 15) are unchanged.

Restrictions

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

Parameter Definitions

EP — The name of the program to be linked to, specified as 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. The file name 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 — The name of the program to be linked. Specified as a byte address, which must not be in a user's code section, at which there is an 8-byte character string giving the name of the member to be concatenated with the current program library name or system library name (as for the EP parameter). This must be specified in a form allowable in the D2(B2) fields of the SS-type assembly instruction format.

LIBRARY — A byte address at which there is an 8-byte character string giving an alternate user program library name for use on this LINK and LINKs nested below this link, or a character string in single quotes giving this name. The previous default library name becomes effective again upon UNLINK to this LINK issuer.

VOLUME — Name of the volume containing an alternate user program library, specified as for the LIBRARY parameter.

SYSTEM

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

NOFAIL

Specifies that the program is not 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 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 topword of the stack to indicate the specific error condition.

LOADONLY

Specifies that after the code section of the new program or subprogram is made addressable, and all initialization of modifiable data areas (including the link return list) is accomplished, control is 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 is in register 0 when control is returned to the LINK issuer. The LINK SVC must be issued from segment 0 if this option is used.

RESTRICT

Allows the access rights assumed by the linked-to program to be restricted to those of the user. The default is NO. The use of this operand does not prevent special rights of a linked-to program from being observed.

XPARENT

Specifying YES makes this link level a transparent link level in respect to satisfying GETPARM requests. Transparent link levels facilitate sending runtime parameter values from PUTPARMs to GETPARMs issued at high link levels. GETPARM requests by a later link level may be satisfied by parameter information supplied by PUTPARMS of an earlier nontransparent link level. If NO is specified, GETPARM requests by a later link level will not be satisfied by parameter information beyond this link level.

PERM

Specifying YES indicates that the program file is to be opened for an indefinite period of time and all system control blocks which are created to control the program file will not be deleted when the program completes execution of the current invocation. NO is the default. A performance enhancement for frequently run programs.

NOPERM

Specifying YES indicates that an indefinitely opened program file is to be reset to remove the indefinitely open status. If the user count of the program file is zero, UNLINK is called to deallocate associated control blocks. NO is the default.

Stack On Input

Four or eight words on top of the stack, as follows:

```
                                         Lower
               |                    |    Address
               |0    1    2    3    |
      0(SP)    |____________________|
               |    |               |
               |(1) | (2) Program   |
               |    |     Name      |
      4(SP)    |____________________|
               |                    |
               |                    |
      8(SP)    |____________________|
               |    |               |
               |    | (3) Unused    |
               |    |               |
     12(SP)    |____________________|
               |                    |
               |                    |
     16(SP)    |____________________|
               | (4) Alternate Volume|
               |     Name           |
     20(SP)    |____________________|
               |          |         |
               |          |(5)      |
               |          |         |
     24(SP)    |____________________|
               | (5) Alternate Library|
               |   Name             |
     28(SP)    |____________________|
               |          |         |
               |          |(6)      |    Higher
               |          |         |    Address
               |____________________|
               |     Preceding      |
               |     Stack Data     |
```

(1) Flag byte:

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 entry point address of new program in register 0. (See LOADONLY operand description of LINK macro in Chapter 4.)

Bit 3 Must be 0.

Bit 4   1 = Alternate user program library and volume are specified in bytes 16-29.
Bits 5-7 Must be 0.

(2) Program name -- string of eight ASCII characters that contains the name of the program to be linked. This name is used with the current program library specification and the resulting file specification is sought. If the file is not found, the file name is used with the system library specification, and the resulting file specification 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.

(3) Unused -- not examined.

(4) Alternate search volume name -- character string that contains overriding user program library's volume serial. Not required unless flag bit 4 is set.

(5) Alternate search library name -- character string that contains overriding user program library. Not required unless flag bit 4 is set.

(6) Not used. Not required unless flag bit 4 is set.

Stack On Entry to Linked to Program

The stack on entry to the linked-to program is as follows:

```
                |                              |     Lower Address
                |______________________________|
      (SP) |  |                              T
                | SAVE AREA FOR 'RTC' TO       |
                |  UNLINK SVC__________________|   68 BYTES
                |                              |
     (R14) |    | STATIC AREA FOR LINKED-TO    |
                | PROGRAM______________________|   VARIABLE
                | LIBRARY, MEMBER, VOLUME      |
                | OF PROCEDURE FILE            |   24 BYTES
                | IF PROCEDURE INTERPRETER     |
                | INITIATED____________________|
                | USER PROGRAM LIBRARY         |
                | AND VOLUME BEFORE LINK       |   16 BYTES
                | (RESTORED BY UNLINK)_________|
                |                              |
                | LINK SVC SAVE AREA___________|   72 BYTES
             |                              T
                | ISSUING PROGRAM'S STACK      |
                | AREA_________________________|   VARIABLE
                |                              |
                |                              |
                                                   Higher Address
```

Stack on Output

```
                                                Lower
              |                      |          Address
              |                      |          Higher
      0(SP) | |______________________|          Address
              |       Preceding      |
              |      Stack Data      |
```

If the specified member cannot be LINKed to, 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

.

Output

A return code is issued in the top word of the stack. The return codes for this macro are as follows:

Code	Description
0	Not a program file, and the procedure interpreter cannot be invoked.
4	Volume not mounted.
8	Volume in exclusive use by another user.
12	All buffers in use when one was required by LINK.
16	Directory not found.
20	File not found.
24	Exceeded allowable link levels.
28	Access to program's file-protection class denied.
32	FDX1 and FDX2 conflict detected by READFDR.
36	FDX2 and FDR conflict detected by READFDR.
40	Invalid parameter passed to READFDR (including NL volume type).
44	I/O error on VTOC.
48	Unable to read FDR2 record (additional extent specifications).

Code	Description
50	Unable to complete static area formation.
52	Invalid program file; unable to complete LINK.
56	File open for other than shared read-only access.
60	Insufficient address space for code section - check user modifiable area size.
64	FDR3 could not be read on a volume set.
68	File has more than one segment.
72	Volume is not mounted.

Examples

```
 LNKTO  LINK EP='SORT'
+LNKTO  PUSHC 0(16,0),*+10
+             B     *+20
+             DC    B'00000000'                FLAGS
+             DC    CL8'SORT'                  NAME
+             DC    XL3'0'                     UNUSED
+             DC    CL4' '                     PASSWORD KEY
+             SVC 4       (LINK)

 LNK1   LINK EP='EDITOR',SYSTEM
+LNK1   PUSHC 0(16,0),*+10
+             B     *+20
+             DC    B'10000000'                FLAGS
+             DC    CL8'EDITOR'                NAME
+             DC    XL3'0'                     UNUSED
+             DC    CL4' '                     PASSWORD KEY
+             SVC 4       (LINK)

 LNK2   LINK EP='MAIL',LIBRARY='MAILLIB',VOLUME='WORK',NOFAIL
+LNK2   PUSHC 0(32,0),*+10
+             B     *+36
+             DC    B'01001000'                FLAGS
+             DC    CL8'MAIL'                  NAME
+             DC    XL3'0'                     UNUSED
+             DC    CL4' '                     PASSWORD KEY
+             DC   CL6'WORK'                   VOLUME
+             DC   CL8'MAILLIB'                LIBRARY
+             DC   XL2'0'                      UNUSED
+             SVC  4  (LINK)
```

4.2.44 LINKPARM - Supply Program Parameters (SVC 33)

Syntax

Format 1:

```
[label] LINKPARM    PUT,{DISPLAY},PRNAME='string',LABEL={(register)}
                        { ENTER }                       { address   }

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

                        {,FMTLIST={(register)}   }
                                  { address  }

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

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

Format 2:

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

Format 3:

```
[label] LINKPARM    REFER{, MERGE }[,REMOVE],FMTLIST={(register)}
                            {,NOMERGE}               {  address }

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

Function

LINKPARM performs the following functions:

- PUT supplies parameters to another program's GETPARMs before issuing the LINK SVC to invoke the other program.

- CLEANUP cleans up the various internal data structures created by the PUT function.

- REFER allows the calling program access to any parameters which the user may have changed at GETPARM time, or to return the address of a previously created and labelled FMTLIST.

Both the PUTPARM macro and the LINKPARM macro call the PUTPARM SVC. The PUTPARM macro allows only the use of parameters of another program (the PUT function), while the LINKPARM macro accesses all the functions of the PUTPARM SVC.

The parameters to be supplied to the GETPARM are contained in a data structure, 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 buffer in the modifiable data area 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 user modifiable 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 are copied to the GETPARM KEYLIST (left-aligned and truncated). To solicit modifications by the user, A GETPARM workstation interaction can be requested by selecting the DISPLAY option; otherwise, a workstation interaction 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 are used in the order in which they were supplied to the PUTPARM SVC. Normally, no two GETPARM requests access the same FMTLIST. FMTLIST may be declared to be for repeated use via the macro parameter REPEAT.

FMTLIST may be labeled for later use through the use of the LABEL parameter. This 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 an FMTLIST (e.g., via the REFERLABEL parameter of the LINKPARM macro), a pointer to the labeled FMTLIST is stored, causing GETPARM to reuse the labeled FMTLIST.

As an example of the backward reference facility, suppose that the program requiring parameters requests the same set of parameters several times and that the calling program is suppressing the workstation interactions. 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 interaction. 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 interaction.

The PUTPARM SVC also supports an override facility. If the prname specified by the linking program matches the LABEL of FMTLIST specified by the linked-to program, the parameter values in the linking program's FMTLIST 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, program 1 issues the following LINKPARM (FMTL1 sets KEY2 to PROG1):

```
    LINKPARM PUT, PRNAME='OVERRIDE',FMTLIST=FMTL1
```

Program 1 then links to program 2. Program 2 issues the following LINKPARM (FMTL2 sets KEY1 and KEY2 to 'PROG2):

```
    LINKPARM PUT,PRNAME='DEMO',LABEL='OVERRIDE',
         FMTLIST=FMTL2
```

Program 2 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 PF key. This may be done in one of two ways, via either the PFKEY or AID parameter. Both can pass the full range of 32 PF keys plus ENTER. PFKEY takes either the actual key number (1-32) or the keyword ENTER. AID takes the AID character of the PF key, where A-P correspond to PF keys 1-16 respectively, a-p correspond to PF keys 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 PF key is passed to GETPARM depends on whether the LINKPARM is a normal or a backward reference. In the normal case, the PF key is placed into the first byte of the FMTLIST addressed by FMTLIST by the LINKPARM macro. The original FMTLIST is modified. In the case of a backward reference, the PF key is placed onto the stack and then into the FMTLIST buffer. The original FMTLIST is not modified in this case.

Parameter Definitions

PUT — Enables a program to supply parameters to a GETPARM issued by another program. The parameters supplied to the GETPARM are contained in a data structure 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 can not use the LINKPARM PUT function to pass parameters to its own GETPARM.

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 is removed. If the CLEANUP option is used, REFERLABEL is the only other parameter supplied.

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

DISPLAY If specified, requests a workstation transaction when the FMTLIST supplied to the linked-to program is accessed.

ENTER If specified, suppresses a workstation transaction when this FMTLIST is accessed. The default is ENTER.

PRNAME A name of up to eight 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 eight alphanumeric characters which identifies a previously labeled FMTLIST. This parameter is used to backward reference a previously created FMTLIST. This backward reference facility allows a program to reuse the (possibly updated) parameters of a labelled FMTLIST. The REFERLABEL parameter is specified as an expression that addresses an 8-byte field which contains the name of the FMTLIST, a register in parentheses that points to an 8-byte field containing the name of the FMTLIST, or as a character string in single quotes which is the name of the FMTLIST. 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 contains the address of the destination FMTLIST.

FMTLIST The address of an FMTLIST created by the FMTLIST macro which is to be used in this operation. The FMTLIST parameter is specified as (1) an expression that addresses a FMTLIST, or (2) as a register in parentheses that contains the address of the FMTLIST. Optionally, the address of KEYLIST+8 may be supplied since a KEYLIST is identical to an FMTLIST with the exception that the first eight bytes of a KEYLIST contain a prname. For the PUT function, REFERLABEL and FMTLIST are mutually exclusive.

AID The AID (Attention ID) character of a PF key passed to the GETPARM. AID characters are A-P (i.e., PF keys 1-16, respectively), a-p (i.e., PF keys 17-32, respectively), and @ (i.e., the ENTER key). The AID parameter is specified as an expression that addresses a 1-byte field which contains the AID character, a register in parentheses that points to a 1-byte field which contains the AID character, or a character string in single quotes which is the AID character. AID and PFKEY are mutually exclusive.

PFKEY

A PF key 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. PFKEY and AID are mutually exclusive.

LABEL

FMTLIST may be labelled for later use by the backward reference and override facilities. A name of up to eight alphanumeric characters is used to label the saved FMTLIST. The LABEL parameter is specified as an expression that addresses an 8-byte field which contains the label, or as a register that points to an 8-byte field which contains the label.

REPEAT

Specifies whether the FMTLIST may be used again. Normally, no two GETPARM requests access the same FMTLIST. A FMTLIST is declared to be for repeated use via this parameter. If REPEAT=NO or the parameter is not coded, the FMTLIST is used only once. If REPEAT=YES, the FMTLIST is used until it is removed. If REPEAT=n, the FMTLIST is used n+1 times (initial use + n repeats). The value of the repeat count can range from 1-32768. Also specified as an expression that addresses a 2-byte binary repeat count or as a register in parentheses pointing to a 2-byte binary repeat count.

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 (of the buffer in the modifiable data area) of the FMTLIST referenced by the REFERLABEL parameter (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.

Stack On Input

```
                                                Lower
             |                          |       Address
             |0    1       2     3      |
   0(SP)     |_____|_______|____________|
             | (1) |  (2)  | (3)        |
             |_____|_______|____________|
   4(SP)     |                          |
             | (4) Address FMTLIST      |
             |__________________________|
   8(SP)     |                          |
             |                          |
             |__________________________|
  12(SP)     |                          |
             | (5) prname               |
             |__________________________|
  16(SP)     |                          |
             |                          |
             |__________________________|
  20(SP)     |                          |
             | (6) LABEL                |       Higher
             |__________________________|       Address
  24(SP)     |                          |
             |                          |
             |__________________________|
             |        Preceding         |
             |       Stack Data         |
```

(1) Flag byte:

```
        Bit 0   1 = DISPLAY type
        Bit 1   1 = Search for a BWR FMTLIST
        Bit 2   1 = Clean up all PRBs and their FMTLISTs
        Bit 3   1 = Merge into user FMTLIST
        Bit 4   1 = Use repeat count
        Bit 5   1 = Cleanup BWRed PRB and FMTLIST only
```

(2) AID character for GETPARM

(3) Repeat count:

```
        X'00' = Never repeat
        X'01'-X'7FFF' = Repeat count
        X'8000' = Repeat indefinitely
```

(4) Address of FMTLIST or backward referenced LABEL. The FMTLIST to be constructed is as follows:

```
                |                              |
         +0     | (a) PF Key    |(b) Number    |
                |     Field     |    of fields |
         +4     |                              |
                | (c) Field Format             |
                |     Control Block  1         |
                |______________________________|
                |                              |
    +4 + BL     | (c) Field Format             |
          1     |     Control Block  2         |
                |______________________________|
                |              .               |
                |              .               |
                |              .               |
                |              .               |
                |              .               |
      n-1       |______________._______________|
  +4 +    BL    |                              |
          i     | (c) Field Format             |
      i=1       |     Control Block N          |
                |                              |
                |______________________________|
```

where BL = length of format control block

(a) A 1-byte receiving field for the corresponding AID character of the program function key received in a user response to a request for selection. This field may be set by a procedure specification of a function key number.

(b) A 1-byte binary count - number of field format control blocks.

(c) Format control block (variable length field). There are two formats for the format control blocks: one for control of the keyword/receiving field pairs, and the other to control the use of embedded text to be displayed. This field is repeated for each field to be displayed in the order they are to be displayed on the workstation screen.

Keyword/Receiving Field Field Format Control Block Structure

```
          DATA STRUCTURE
        |0    1    2    3    |
        |     |    |    |    |     Lower
        | (1) | (2)| (3)| (4)|     address
        |_____|____|____|____|
        |                    |
        | (5)  Keyword       |
        |____________________|
        |                    |
        | (6)  Receiving     |
        |      Field         |     Higher
         --------------------      address
```

(1) Line-advance-count for display control. 1-byte binary field.

(2) Space-advance-count for display control. 1-byte binary field. Line advance takes place before space advance. Both take place before display of keyword and receiving field.

(3) Field error flag and receiving field entry restriction indicator. 1-byte binary field.

Bit 0: Field error flag:

1 = Error - set by program to draw attention to fields in error. Reset by GETPARM.

Bits 5-7: Receiving field 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. An all blank response is 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 are ignored. An all blank response is treated as a legitimate NULL response. Field length is restricted to 16 characters.

4 = Uppercase alphanumeric. All entered letters are converted to uppercase. Legal nonblank response must be left-justified and consist entirely of the numerals 0-9, the letters A-Z, the special characters (@, #, or$), and trailing blanks. An all blank response is treated as a legal NULL response indicator. Maximum usable field length is 68 characters.

5 = Uppercase hexadecimal. All entered letters are converted to uppercase. 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. An all blank response is 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 are converted to uppercase. Legal nonblank responses are left-justified, beginning with a letter from A-Z, or one of the special characters (@, #, or $), and consist entirely of the numerals 0-9, the letters A-Z, the special characters, and trailing blanks. All blank responses are treated as a legal NULL response indicator. Maximum usable field length is 68 characters.

(4) A 1-byte binary receiving field length minus one (in characters).

(5) An 8-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.

Embedded Text Field Format Control Block Structure

```
      DATA STRUCTURE
 |0    1    2    3      |
 |     |     |    |     |      Lower
 | (1)|  (2) |(3)|  (4) |      address
 |     |     |    |     |
 |______________________|
 |                      |
 | (5) Text             |
 |                      |
 |______________________|      Higher
                               address
```

(1) Line-advance-count for display control. 1-byte binary field.

(2) Space-advance-count for display control. 1-byte binary field. Line advance takes place before space advance. Both take place before display of keyword and receiving field.

(3) The value "-1" (=255).

(4) Text field character length minus 1. 1-byte binary field.

(5) Character string to be displayed. Variable length field.

Stack On Output

```
                                               Lower
                 |                      |      Address
                 |______________________|
      0(SP)      |                      |
                 |      Return Code     |
                 |______________________|
      4(SP)      |                      |
                 |    FMTLIST Address   |      Higher
                 |____of this PRB_______|      Address
                 |    Preceding         |
                 |   Stack Data         |
```

Return Codes

Code	Description
0	Success.
2	Backward Reference Error.
8	Bad FMTLIST.
12	PRB error.
16	CLEAN-PARM error.
20	MERGE-PARM error.

Example

```
 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    0,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)
```

4.2.45 LNKB - Describe Link Return List Block

Syntax

```
    LNKB     [NODSECT][,REG=expression][,SUFFIX=character][,PREFIX={NO }]
                                                                {YES}
```

Function

Describes the link return block (LNKB) used with the LINK SVC. It is in the format of an SVC or JSCI stack item, and can be extended to contain information for procedure interpretation and debugging.

Parameter Definitions

NODSECT Specification of NODSECT results in the LNKB fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, a DSECT with the name LNKB (plus optional suffix) is generated.

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

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

PREFIX IF YES, the structure is extended to include the following information: library, file and volume of procedure to be run.

Strucutre

		BYTE 0	BYTE 1	BYTE 2	BYTE 3
LNKB: START					
	+0	PROCLIB			
	+4				
	+8	PROCFIL			
	+C				
	+10	PROCVOL			
	+14			SPARE1	
	+18	SAVEVOL			
	+1C			SAVELIB	
	+20				
	+24			SPARE2	
REGSV	+28	REG0			
	+2C	REG1			
	+30	REG2			
	+34	REG3			
	+38	REG4			
	+3C	REG5			
	+40	REG6			
	+44	REG7			
	+48	REG8			
	+4C	REG9			

		BYTE 0	BYTE 1	BYTE 2	BYTE 3
	+50	REGA			
	+54	REGB			
	+58	REGC			
	+5C	REGD			
	+60	REGE			
TYPE	+64	CALLCHN			
RTNPTR	+68	OPCW			
	+6C				
	+70	FLAG	PROGNAME		
	+74				
	+78		SPARE3		
	+7C				
	+80	PROGVOL			
	+84			PROGLIB	
	+88				
	+8C			SPARE4	

Example

```
         LNKB NODSECT
*,* LNKB DEFINITION
*
* THE LINK RETURN LIST BLOCK (LNKB) IS ENSTACKED ON A 'LINK' SVC AND
* LOCATED THROUGH THE PFB CHAIN, ROOTED IN ETCBPFB. IT IS IN THE
* FORMAT OF AN SVC OR JSCI STACK ITEM, AND MAY BE EXTENDED
* TO CONTAIN INFORMATION FOR PROCEDURE INTERPRETATION AND DEBUGGING.
*
*        DATE 3/09/83
*        RELEASE 5.04
*
LNKBBEGIN                 EQU *          START OF SVC/JSI SAVE AREA
LNKBREGSV                 DS  15F        REGISTERS 0-14 SAVE AREA
                          ORG LNKBREGSV
LNKBREG0                  DS  F          ... REG0 SAVE
LNKBREG1                  DS  F          ... REG1 SAVE
LNKBREG2                  DS  F          ... REG2 SAVE
LNKBREG3                  DS  F          ... REG3 SAVE
LNKBREG4                  DS  F          ... REG4 SAVE
LNKBREG5                  DS  F          ... REG5 SAVE
LNKBREG6                  DS  F          ... REG6 SAVE
LNKBREG7                  DS  F          ... REG7 SAVE
LNKBREG8                  DS  F          ... REG8 SAVE
LNKBREG9                  DS  F          ... REG9 SAVE
```

```
LNKBREGA                   DS  F           ... REGA SAVE
LNKBREGB                   DS  F           ... REGB SAVE
LNKBREGC                   DS  F           ... REGC SAVE
LNKBREGD                   DS  F           ... REGD SAVE
LNKBREGE                   DS  F           ... REGE SAVE
LNKBTYPE                   DS  0BL1        TYPE OF CALL ('SVC' IF LINK)
LNKBTYPESVC                EQU X'01'       ... (SVC)
LNKBTYPEJSCI               EQU X'00'       ... (JSCI)
LNKBFLAGS                  DS  0BL1        Flags (expansion on type)02\
LNKBFLAGSSTAND             EQU X'80'       Standard frame           02\
LNKBFLAGSEX                EQU X'40'       Created by exception hand02\
LNKBFLAGSMOD               EQU X'20'       Module frame             02\
*                          EQU X'10'       Reserved                 02\
LNKBFLAGSLEVEL             EQU X'0E'       Proc Level (B'00001110') 02\
LNKBFLAGSSVC               EQU X'01'       SVC flag                 02\
LNKBCALLCHN                DS  A           CALL/LINK/SVC BACK CHAIN
LNKBMASK                   DS  0BL1        Program mask             02\
LNKBRTNPTR                 DS  A           ADDRESS TO RETURN CONTROL
*
LNKBJSILENGTH              EQU *-LNKBBEGIN
*
                           ORG LNKBRTNPTR
LNKBOPCW                   DS  BL8         PCW SAVE AREA (LINK SVC)
*
LNKBSVCLENGTH              EQU *-LNKBBEGIN
*
LNKBFLAG                   DS  BL1         FLAGS FOR LINK SVC:
LNKBFLAGSYS                EQU X'80'       . SEARCH SYSTEM DIRECTORY
*                                            ONLY
LNKBFLAGNOFAIL             EQU X'40'       . RETURN TO INVOKER AT NEXT
*                                            INSTRUCTION ADDRESS + 4 IF
*                                            'LINK' FAILS (RETURN CODE
*                                            IN TOP WORD OF STACK)
LNKBFLAGLOAD               EQU X'20'       . RETURN TO INVOKER AT NEXT
*                                            INSTRUCTION ADDRESS + 8
*                                            INSTEAD OF INVOKING NEW
*                                            PROGRAM OR PROCEDURE
*                                            (ENTRY POINT ADDRESS
*                                            RETURNED IN R0)
LNKBFLAGREST               EQU X'10'       . Restrict Access Rights
*                                            to Logon Rights
LNKBFLAGLLV                EQU X'08'       . LINK LIBRARY/VOLUME
*                                            SUPPLIED
LNKBFLAGSUBP               EQU X'04'       . LINK TO 'SUBPROGRAM'
LNKBFLAGXPARENT            EQU X'02'       . "TRANSPARENT LINK" 04\
LNKBPROGNAME               DS  CL8         NAME OF PROGRAM OR
*                                          PROCEDURE TO BE RUN/LOADED
LNKBSPARE3                 DS  BL7         (UNUSED)
LNKBSEND                   EQU *
LNKBPROGVOL                DS  CL6         OPTIONAL LINK VOLUME
```

```
LNKBPROGLIB                 DS  CL8         OPTIONAL LINK LIBRARY
LNKBSPARE4                  DS  BL2         (UNUSED)
LNKBEND                     EQU *
LNKBLENGTH                  EQU LNKBEND-LNKBBEGIN
```

4.2.46 LOADCODE - Load Microcode for Devices/IOPs (SVC 45)

Syntax

```
[label] LOADCODE    FUNCTION={DEVICE     },MCID={ address  },
                             {PERIPHERAL }      {(register)}
                             {CONFIGTABLE}
                             {(register) }

                    DEVICE={ address  },START={ address  },
                           {(register)}       {(register)}

                    LENGTH={ address  },INTERRUPT={YES},
                           {(register)}           {NO }

                    PLIST=(SP),CLOAD={YES},UNLOAD={YES},
                                     {NO }        {NO }

                    MCFILE={(register)},MCLIB={(register)},
                           { address  }       { address  }

                    MCVOL={(register)}[,MEMA={(register)},
                          { address  }       { address  }

                    MCOPTS=({RENEWMC },{SYSUINT })
                            {NRENEWMC} {TASKUINT}
```

Function

Calls the LOADCODE SVC to load microcode into a programmable IOP or a device.

NOTE

The system maintains an 8-character field which names the library in which all default microcode files reside. This library, which is set during the SYSGEN process, must be on the system volume. The default library name is @SYSTEM@.

Restrictions

An unprivileged caller can load microcode to a peripheral processor (data link processor) if that peripheral processor is exclusively reserved by the caller. LOADCODE returns a failure code if the peripheral processor is not reserved by the caller, or is reserved by another task.

Parameter Definitions

FUNCTION Coded as one of the following options:

- DEVICE -- Used to load code into a device such as a workstation.
- PERIPHERAL -- Used to load code into a peripheral processor (PP).
- CONFIGTABLE -- Used to load a device configuration table.

FUNCTION can also be coded as a register in parentheses containing a value. This parameter is required unless PLIST=(SP).

MCID The type of microcode to be loaded. Specified as an address expression pointing to a 1-byte field, or a register in parentheses containing the value. Required unless either INTERRUPT=YES or PLIST=(SP).

DEVICE The VS device number for the device where code is to be loaded. Specified as an address expression pointing to a 1-byte field containing a value from 0 - 255, or a register in parentheses containing the value. Required unless PLIST=(SP).

START The address destination in the specified device where code is to be loaded. Specified as an address expression pointing to a 4-byte field, or a register in parentheses containing the address. Required unless either INTERRUPT=YES or PLIST=(SP).

LENGTH The length of the code which is to be loaded. Specified as an address expression pointing to a 4-byte field, or a register in parentheses containing the length in bytes. Required unless either INTERRUPT=YES or PLIST=(SP). Also not required if the load-by-name option is used (i.e., MCFILE, MCLIB, and MCVOL are supplied).

INTERRUPT Coded either YES or NO as shown. Indicates that the microcode type is taken from the system entry for the particular DEVICE. Optional; the default is NO.

PLIST Indicates that the parameter list is already on the stack. Code PLIST=(SP) only, since the LOADCODE SVC only accepts parameters on the stack. If supplied, all other parameters are optional.

CLOAD

If CLOAD=YES, microcode is conditionally loaded to a peripheral processor or device; that is, loading occurs only if the currently active microcode type is not the same as the microcode type-ID provided in the MCID parameter. If these microcode types are identical, LOADCODE returns a return code that indicates success. If these microcode types are not identical, a normal LOADCODE results. The default is NO.

UNLOAD

If UNLOAD=YES, LOADCODE reloads the microcode type associated with a peripheral processor or programmable device. The UNLOAD function reloads this default microcode type into the device or peripheral processor (this option can be combined with "conditional load", so that the LOADCODE is only done if the current microcode type differs from the default microcode type). The default is to NO.

MCFILE
MCLIB
MCVOL

These parameters allow both unprivileged and privileged callers to load-by-name, that is, to specify a file name, a library name, and a volume name from which LOADCODE is to read the microcode file. The file name (MCFILE) is always required if a "load by name" is to be done; the library name (MCLIB) and volume name (MCVOL) are optional and default to the system microcode library and volume. Volume specification is ignored if the library is not specified.

MCFILE and MCLIB are specified as a register in parentheses pointing to an 8-byte field that contains the file or library name, or as an expression that addresses an 8-byte field containing the file or library name. MCVOL is specified as a register in parentheses that points to a 6-byte field which contains the volume name, or as an expression that addresses a 6-byte field which contains the volume name.

This option is incompatible with the UNLOAD option.

MEMA

Specifies the starting address of a memory-resident microcode program to be loaded to a device.

MCOPTS

If RENEWMC is specified, the microcode is reloadable on an interrupt-driven call. If NRENEWMC is specified, the microcode is not reloadable on an interrupt-driven call; any error completion with the data link processor or the peripheral processor is passed back to the XIO issuer. The default is RENEWMC.

If SYSUINT is specified, unsolicited interrupts are handled by the system. If TASKUINT is specified, all unsolicited interrupts are handled by the issuing task (including power-on and HELP interrupts). The default is SYSUINT.

Stack On Input

Three or nine words are on top of the stack, as follows:

```
                                            Lower
                |                 |         Address
                |0   1    2    3  |
        0(SP)   |___________________|
                | (1) | (2) | (3) | (4)|
                |     |     |     |    |
        4(SP)   |___________________|
                | (5) Start Address |
                |                   |
        8(SP)   |___________________|
                | (6) Microcode Length |
                |                   |
       12(SP)   |___________________|
                | (7) File Name of  |
                |     User Microcode|
       16(SP)   |___________________|
                |                   |
                |                   |
       20(SP)   |___________________|
                | (8) Library Name of |
                |     Microcode File|
       24(SP)   |___________________|
                |                   |
                |                   |
       28(SP)   |___________________|
                | (9) Volume Name of |
                |     Microcode File|
       32(SP)   |___________________|
                |         | (10)    |
                |         |         |
                |___________________|
                |     Preceding     |   Higher
                |     Stack Data    |   Address
```

(1) Option flags (byte 0):

X'01' Load device.
X'02' Load peripheral processor (PP or IOP/IOC).
X'04' Load configuration table.
X'08' Load device routing table.
X'10' Unload to the default microcode file.
X'20' Load by name (Microcode file-library-volume).
X'40' Conditional load.
X'80' Interrupt-driven entry - take file name from control blocks if set, otherwise the file name is taken from the input.

(2) Microcode type ID (byte 1)

(3) Microcode option byte (byte 2)
X'80': Nonrenewable microcode -- do not reload on interrupt-driven call. Any error completion with PP is passed back to the XIO issuer.
X'40': For workstations -- task will handle all interrupts (including power-on and HELP).

(4) Device number (byte 3)

(5) Start loading address of the device microcode (bytes 4-7)

(6) Length of the microcode to be loaded (bytes 8-11)

If X'20' (Load by name) is set in the FLAG byte, the following parameters are expected on the stack:

(7) File name in ASCII of private microcode file (CL8) (bytes 12-19) If byte twelve is equal to X'80', then bytes 13-15 are the starting memory address of the microcode to be loaded.

(8) Library name in ASCII for private microcode file (CL8) -- the system microcode library and volume are used if this field is XL8'00' (bytes 20-27).

(9) Volume name in ASCII for private microcode file (CL6) -- the system microcode volume is used if this field is XL6'00'. Note that a microcode library must be provided if this field is to be referenced (bytes 28-33).

(10) Reserved, must be 0 (bytes 34-35).

Stack On Output

```
                                          Lower
           |                      |       Address
           |______________________|
   0(SP)   |                      |
           |    Return Code       |       Higher
           |______________________|       Address
           |      Preceding       |
           |     Stack Data       |
```

Output

LOADCODE replaces the input parameters on the stack with a word indicating success or failure of the operation.

Return Codes

Code	Description
0	Success.
4	Device or peripheral processor specified is not programmable.
8	Specified microcode file not found. Also set when the specified class and type of microcode is not included in the UCB MC list, or when the specified file name is not a valid alphanumeric string.
12	Device or peripheral processor not exclusively reserved by the caller.
16	Error in opening microcode file, or file not consecutive.
20	I/O error when reading microcode file.
24	(a) I/O error while loading device microcode, peripheral processor (PP) microcode, or configuration tables. (b) Error when restarting device or peripheral processor (PP) after loading microcode. (c) Unable to load device because peripheral processor (PP) code is missing, or attempt to load peripheral processor (PP) fails for any reason. (d) Unable to load peripheral processor (PP) code because configuration tables are missing, or attempt to load tables fails for any reason.
28	Insufficient memory pool (GETMEM failure).
32	Reserved.
36	Incompatible options: (a) UNLOAD and load-by-name both specified. (b) CLOAD and INTERRUPT both specified.
40	Other devices on cluster not all reserved by the calling task (noninterrupt-driven LOADCODE only).
44	Forced cancel signal received (loadcode incomplete).
48	Unprivileged caller.
52	Device on multi workstation cluster busy.
56	No GETHEAP space on attempt to save multi workstation screens.

Examples

```
        LOADCODE FUNCTION=DEVICE,MCID=UCBMCTYPE,DEVICE=UCBADDR,
        START=(R8),LENGTH=(R9)
+       DS         0H
+       PUSH       0,R9                  MICROCODE LENGTH
+       PUSH       0,R8                  START ADDRESS
+       PUSHA      0,0                   SPACE FOR MORE PARAMETERS
+       MVI        0(15),4               FUNCTION FLAG
+       MVC        3(1,15),UCBADDR       DEVICE NUMBER
+       MVC        1(1,15),UCBMCTYPE     MICROCODE TYPE ID
+       SVC        45   (LOADCODE)

        LOADCODE FUNCTION=DEVICE,INTERRUPT=YES,DEVICE=UCBADDR,
        PLIST=(SP)
+       DS         0H
+       MVI        0(15),129             FUNCTION FLAG
+       MVC        3(1,15),UCBADDR       DEVICE NUMBER
+       SVC        45   (LOADCODE)
        LOADCODE PLIST=(SP)
+       DS         0H
+       SVC        45   (LOADCODE)


 INIT   LOADCODE FUNCTION=DEVICE,MCID=(R2),DEVICE=DEVADR,START=(R5),  -
        LENGTH=CODLTH
+INIT   DS         0H
+       PUSHC      0(4,0),CODLTH         Microcode Length
+       PUSH       0,R5                  Start Address
+       PUSHA      0,0                   Space For More Parameters
+       MVI        0(15),1               Function Flag
+       MVC        3(1,15),DEVADR        Device Number
+       STC        R2,1(,15)             Microcode Type ID
+       SVC        45 (LOADCODE)
```

4.2.47 LOCAL - Generate Local Symbols

Syntax

```
[label] LOCAL   [count,...][,PREFIX={LCL#  }]
                                    {string}
```

Function

Automatic generation of unique local symbol names.

Parameter Definitions

count — Specifies on which local symbol to generate a local symbol name. An integer from 0 to 9 may be specified and the parameter can be repeated as necessary.

PREFIX — A character string used to generate the local symbol name. If not supplied, the macro uses the character string LCL#.

Example

```
         MACRO
         LOAD
         COPY LOCALS
         GBLC  &LOCAL0,&LOCAL1,&LOCAL2,&LOCAL3,&LOCAL4
         GBLC  &LOCAL5,&LOCAL6,&LOCAL7,&LOCAL8,&LOCAL9
         LOCAL 0,1,3,PREFIX=A
&LOCAL0  LR    R1,R2
&LOCAL1  LR    R3,R4
&LOCAL2  LR    R5,R6
&LOCAL3  LR    R7,R8
         ENDLOCAL
         MEND
```

```
*
                .
                .
                .
                LOAD
+A0             LR     R1,R2
+A1             LR     R3,R4
+               LR     R5,R6
+A2             LR     R7,R8
                LOAD
+A3             LR     R1,R2
+A4             LR     R3,R4
+               LR     R5,R6
+A5             LR     R7,R8
                LOAD
+A6             LR     R1,R2
+A7             LR     R3,R4
+               LR     R5,R6
+A8             LR     R7,R8
```

4.2.48 LOGOFF - Log Off Interactive Terminal (SVC 43)

Syntax

```
[label] LOGOFF
```

Function

LOGOFF terminates a task by request of an active program. LOGOFF marks the task as being in logoff procedures and having no debugging privileges and then issues a CANCEL SVC with a message number of 0001, LOGOFF (SVC)43). LOGOFF returns to the calling program with error message 0002, "Invalid parameter list passed to LOGOFF", if the two words passed to the SVC do not contain binary zeroes.

Stack On Input

```
                                         Lower
              |                       |  Address
              |_______________________|
      0(SP) | |                       |
              |     Must be zero      |
              |_______________________|
      4(SP)   |                       |
              |     Must be zero      |  Higher
              |_______________________|  Address
              |       Preceding       |
              |      Stack Data       |
```

Stack On Output

```
                                         Lower
              |                       |  Address
              |_______________________|
      0(SP) | |       Preceding       |  Higher
              |      Stack Data       |  Address
```

Example

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

4.2.49 MOUNT - Mount Disk or Tape Volume (SVC 30)

Syntax

Format 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 }][,GENERIC={NO }]
                     {F}         {YES}        {YES}           {YES}

           [,NSA={NO }][,NODISPLAY={NO }][,PAGING={NO }]
                 {YES}             {YES}          {YES}

           [,NOMESSAGE={NO }][,SECURE={NO }][,VSID={(register)}]
                       {YES}          {YES}        {  integer }
                                                   {  address }
```

Format 2:

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

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

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

Function

Mount performs the following functions:

- Requests a disk volume to be mounted on the indicated device with the specified label, usage, type, spool file, and work file attributes.

- Requests a tape volume to be mounted on the indicated device with the specified label attributes.

To perform a disk or tape mount operation, the input parameters are first validated, and if correct, a mount message is displayed on the user's workstation to mount the proper volume. If the NODISPLAY option is chosen, the message appears only on Workstation 0. When the volume is mounted and the device is ready, the new volume label will be read and checked. The volume control block is updated with the information.

The NOMESSAGE option indicates that the volume to be mounted is already on the disk or tape drive. No mount message is displayed, and the VCB information is updated from that volume label.

The Bypass-label-processing option (BLP) is used by the disk or tape initialization program and the floppy copy program (FLOPYDUP).

NOTE

A nonstandard 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 nonstandard addressing is specified, the XIO SVC does not perform extent validation or address translation, but simply passes the address to the firmware via the I/O control word (IOCW).

Volume Set Mounts

The MOUNT SVC also allows for mounting volumes that are part of a volume set. A volume set may have from 1 to 255 different volumes, each volume having the same volume name but its own volume set identification number (VSID) which is created with the DISKINIT utility.

When mounting such a volume, or a single volume, on a specific device, only the volume name is required and any volume provided with the same name will satisfy the request. If you specify the VSID parameter, then the volume you mount must match the name and VSID combination. On a GENERIC mount, you must provide the VSID and the VOLUME name in the call to the MOUNT SVC.

Not all volumes in a volume set need to be mounted at the same time. However for file access, the root volume (VSID=1) must be mounted as it contains the master directory for all files within the set. Each subsequent MOUNT command issued for a volume in the set must specify the same USAGE parameter value as specified on the first MOUNT. To change the USAGE parameter values for a set which is mounted, perform a remount of the root volume.

Parameter Definitions

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 1-byte field containing the device number in binary. One of these parameters is required and they are mutually exclusive.

VOLUME The name of the volume which is to be mounted. The name can be specified as a register in parentheses that points to the volume name, as a character string in single quotes which is the volume name, or as an expression that addresses a 6-byte field which contains the volume name. This parameter is required.

BLP This parameter instructs the system to bypass label processing and checking and should be specified with care. Valid values are YES and NO. The default is 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 SL. For a tape volume the default is AL. SL is valid for disk volumes only; AL and IL are valid for tape volumes only.

USAGE Denotes volume access and dismounting restrictions. Dismounting restrictions also apply to remounting with different attributes. Valid values are

- SH -- Shared, the volume may be accessed and dismounted by any user. Default for disk and tape volumes.

- RR -- Restricted removal, the volume may be accessed by any user but dismounted only by the user who mounted the volume. For disk volumes only.

- PR -- Protected, files on the volume may be read by any user but updated and dismounted only by the user who mounted the volume. For disk volumes only.

- EX -- Exclusive, the volume can be accessed and dismounted only by the user who mounted the volume.

For volume sets that have more than one member volume mounted, these values may be changed by doing a remount of the root volume.

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

SPOOL YES denotes that the volume is included in the list of volumes scanned when the system creates a spool (print) file for a user. The default is NO. This parameter is valid for disk volumes only.

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

NSA If YES is specified, indicates that the volume to be mounted follows nonstandard addressing conventions. The default is NO.

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

NOMESSAGE If YES is specified, indicates that the volume to be mounted is already on the disk or tape drive. No mount message is displayed, and the volume control block (VCB) information is updated from the volume label. The default is NO.

PAGING If YES is specified, indicates whether a standard label disk volume accepts paging files. Only valid for standard label volumes.

SECURE Allows the system to observe special program privileges for programs residing on the disk. Only security administrators can set this option.

GENERIC YES specifies that the volume can be mounted on any appropriate available drive. NO specifies that the volume is to be mounted on the device specified by the DISK parameter. Specifying GENERIC is not valid for tape drives.

VSID Specifies the volume set identification number for the volume to be mounted. The number ranges from 0 to 255. A single volume has a VSID of 0. Not valid for tape mounts and not required for single volumes.

Stack On Input

The parameter list is either eight or sixteen bytes long depending upon whether the high bit of the third byte is set.

```
               |                      |    Lower
               |                      |    Address
               |0     1      2     3  |
   0(SP)  |    |______|______|________|
               | (1)  | (2)  | (3)    |
               |      |      |        |
   4(SP)       |______|______|________|
               |                      |
               |                      |
   8(SP)       |______________________|
               | (4)  |  (5) | (6)    |
               |      |      |        |
  12(SP)       |______|______|________|
               |                      |    Higher
               |                      |    Address
               |______________________|
               |      Preceding       |
               |     Stack Data       |
```

(1) Flag byte
For disk volume:

Bit 0	1 = Mount an unlabelled volume.
Bits 1-2	0 = Mount for shared use. 1 = Mount with restricted removal. 2 = Mount with protected use, and restricted removal. 3 = Mount for exclusive use.
Bit 3	0 = Mount a removable volume. 1 = Mount a fixed volume.
Bit 4	1 = No message option. Mount volume on drive.
Bit 5	1 = Mount volume for bypass-label-processing.
Bit 6	1 = Volume allows spool files.
Bit 7	1 = Volume allows work files.

For tape volume:

Bit 0	1 = Mount an unlabelled volume.
Bit 1	0 = Mount for shared use. 1 = Mount for exclusive use.
Bit 2	Unused.
Bit 3	0 = Mount an ANSI tape. 1 = Mount an IBM tape.
Bit 4	1 = No message option. Mount volume on drive.
Bit 5	1 = Mount volume for bypass-label-processing.
Bits 6-7	Unused.

(2) Device number. Binary value in the range of 0 to 255.

(3) Volume name. 6-character volume name. If the high bit of the first byte of this field is set, an extension to the parameter list is included (bytes 8 - 15).

(4) Parameter list extension.

Bit 0	1 = Nonstandard addressing in effect (for soft sectored diskettes only).
Bit 1	1 = No display option: do not display message on user's workstation.
Bit 2	1 = Volume is eligible for paging files.
Bit 3	1 = Special program rights are observed.
Bit 4	1 = VSID is required.
Bits 5-6	Unused, must be zero.
Bit 7	1 = Generic mount - volume may be mounted on any available drive.

(5) Volume set identification number (VSID) - a binary number from 0 to 255.

(6) Unused, must be 0.

Stack On Output

```
                                          Lower
           |                      |       Address
           |______________________|
  0(SP) |  |                      |
           |     Return Code      |       Higher
           |______________________|       Address
           |      Preceding       |
           |     Stack Data       |
```

Output

Return codes 16-84 are set without the mount message being shown on the workstation.

Return Codes

Code Description

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.

16 Device is not a disk or a tape, or the device number is not valid.

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 Single volume already mounted.

40 The volume is currently in use by the operating system or a user.

44 The currently mounted volume is reserved by another user for exclusive use.

48 Insufficient I/O buffer space to perform the mount.

52 GETMEM pool failure. Unable to allocate space for tape I/O control blocks.

Code	Description
56	Invalid request: work or spool filing or both requested in a nonlabelled volume.
60	Invalid request: nonstandard 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 nonstandard addressing request.
76	Wrong addressing mode: caller requested MOUNT for standard addressing but diskette is nonstandard addressing.
80	Device reserved by another user.
84	Mount failed; aborted by user or operator request.
88	Tape drive does not support the requested density.
92	Success, but could not scratch paging library.
96	Cannot use protected volume for paging.
100	User not authorized for "SECURE" function.
116	Volume set member with this VSID already mounted.
124	SUCCESS but volume identification doesn't match.
128	GENERIC mount is not valid on volume mounted for initialization.

Code	Description
132	Request parameter not valid with GENERIC mount.
136	No switch allowed on non-root set members.
140	Generic mount requires VSID.
144	Device must be reserved, set is reserved.
148	Volume set is not reserved, disk on reserved device cannot be mounted as part of the set.
152	NOVTOC volumes cannot be members of volume sets.

Examples

```
 LAB0     MOUNT DISK=(R1),VOLUME='SYSTEM',LABEL=SL,USAGE=SH,VOLTYPE=F,  X
                   SPOOL=NO,WORK=YES
+LAB0     DS    0H
+         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'00010000'  SET FLAGS
+         SVC   30   (MOUNT)       ISSUE SVC

 LAB1     MOUNT DISK=DISKVOL,VOLUME=(R4)
+LAB1     DS    0H
+         PUSHN 0,8                GET TWO WORDS 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

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

```
          MOUNT DISK=(R1),VOLUME='IRSSET',USAGE=SH,VSID=1
+         PUSHN 0,8                 GET 8 BYTES FOR PLIST EXTENSION
+         XC    0(8,15),0(15)       CLEAR PLIST EXTENSION
+         MVI   9(15),1
+         MVI   0(15),B'00001000'   SET EXTENSION FLAGS
+         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'IRSSET'         VOLUME NAME
+         OI    2(15),X'80'         SET 'PLIST EXTENSION' FLAG
+         MVI   0(15),B'00000000'   SET FLAGS
+         SVC   30    (MOUNT)       ISSUE SVC

          MOUNT DISK=(R2),VOLUME='IRSSET',USAGE=SH,VSID=2
+         PUSHN 0,8                 GET 8 BYTES FOR PLIST EXTENSION
+         XC    0(8,15),0(15)       CLEAR PLIST EXTENSION
+         MVI   9(15),2
+         MVI   0(15),B'00001000'   SET EXTENSION FLAGS
+         PUSHN 0,8                 GET TWO WORDS ON THE STACK
+         STC   R2,1(,15)           SET DEVICE NUMBER
+         MVC   2(6,15),*+10        SET VOLUME NAME
+         B     *+10                BRANCH AROUND CONSTANT
+         DC    CL6'IRSSET'         VOLUME NAME
+         OI    2(15),X'80'         SET 'PLIST EXTENSION' FLAG
+         MVI   0(15),B'00000000'   SET FLAGS
+         SVC   30    (MOUNT)       ISSUE SVC
          END BEGIN
```

```
      MOUNT DISK=5,VOLUME='IRRSET',USAGE=SH,VSID=1
+     PUSHN 0,8                 GET 8 BYTES FOR PLIST EXTENSION
+     XC    0(8,15),0(15)       CLEAR PLIST EXTENSION
+     MVI   9(15),1                                            05\
+     MVI   0(15),B'00001000'   SET EXTENSION FLAGS            05\
+     PUSHN 0,8                 GET TWO WORDS ON THE STACK
+     MVI   1(15),5             SET DEVICE NUMBER
+     MVC   2(6,15),*+10        SET VOLUME NAME
+     B     *+10                BRANCH AROUND CONSTANT
+     DC    CL6'IRRSET'         VOLUME NAME
+     OI    2(15),X'80'         SET 'PLIST EXTENSION' FLAG
+     MVI   0(15),B'00000000'   SET FLAGS
+     SVC   30    (MOUNT)       ISSUE SVC
      END BEGIN
      MOUNT VOLUME='IRSSET',USAGE=SH,GENERIC=YES
+     PUSHN 0,8                  GET 8 BYTES FOR PLIST EXTENSION
+     XC    0(8,15),0(15)        CLEAR PLIST EXTENSION
+     MVI   0(15),B'00000001'    SET EXTENSION FLAGS
+     PUSHN 0,8                  GET TWO WORDS ON THE STACK
+     MVC   2(6,15),*+10         SET VOLUME NAME
+     B     *+10                 BRANCH AROUND CONSTANT
+     DC    CL6'IRSSET'          VOLUME NAME
+     OI    2(15),X'80'          SET 'PLIST EXTENSION' FLAG
+     MVI   0(15),B'00000000'    SET FLAGS
+     SVC   30   (MOUNT)         ISSUE SVC
```

4.2.50 MSGLIST - Generate Display Message

Syntax

```
[label] MSGLIST msg#,issuer,'message-segment1'[,'message-segment2'...]
```

Function

Generates a data structure suitable for use with the MSG parameter of the GETPARM and CANCEL macroinstructions.

Restrictions

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

Parameter Definitions

msg# Up to four alphanumeric characters enclosed in single quotes, normally a message number, to be displayed with the message. The message number 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' Message text in single quotes (apostrophes) to be displayed on a single line. Message text can be repeated as often as required to define additional lines to be displayed. No line can be over 79 characters long. The message can contain single quotes. 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'
```

4.2.51 OPEN - Open a File (SVC 0)

Syntax

```
[label] OPEN     UFB={(register)}[,MODE={OUTPUT}][,NODISPLAY]
                     { address  }       {INPUT }
                                        {IO    }
                                        {EXTEND}
                                        {SHARED}

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

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

Function

Prepares a file for processing by Data Management System (DMS) 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.

Opens a new or existing file for I/O processing by the DMS routines. Information from the file's file descriptor record (FDR1), located in the VTOC, is brought into memory. Devices and volumes are allocated and reserved as required. System control blocks, used to keep track of file information, are created. Buffer space is allocated in the user program's modifiable code area. The OPEN function may issue a GETPARM call to interact with the user at a workstation to request information (such as the file, library or volume name, retain days, etc.) which has not been supplied or to correct the information.

Before opening a file, a data structure called a user file block (UFB) must exist for the file. The UFBGEN macroinstruction can be used to create the UFB and to initialize UFB fields. The following information in the UFB must be initialized depending upon the case:

- New and existing files
  - Parameter reference name (UFBPRNAME)
  - Options (UFBF1, UFBF4)
  - Mode (UFBF2)
  - Device class (UFBDEVCLASS)
  - Device dependant flags (UFBF4)
  - OPEN modifier (UFBDXOM) (if DMS/TX section exists)

- New files (mode = OUTPUT)
  - File organization (UFBFORG)
  - Logical record size (UFBRECSIZE)

- New magnetic tape files
  - Physical block size (UFBBLKSIZE)
  - Tape sequence number (UFBTSEQ)
  - Logical record size (UFBRECSIZE)

- New disk file
  - Disk space requirement (UFBNRECS)
  - File organization (UFBFORG)
  - Logical record size (UFBRECSIZE)
  - Compressed record option (UFBFLAGSCOMP)

- New indexed disk file
  - Key position (UFBKEYPOS)
  - Key size (UFBKEYSIZE)
  - Data/index block packing % (UFBPKD,UFBPKI)
  - Alternate keys defined (UFBALTCNT)
  - Recovery blocks (UFBDXRECBLK)

- New alternate indexed disk files
  - AXD1 block address (UFBALTPTR)
  - Alternate key definitions (AXD1)

- New or existing file may optionally contain
  - Complete actual file name and volume (UFBVOLSER, UFBDIRNAME, UFBFILENAME)
  - Device number
  - Unqualified name of member within library or volume (UFBFILENAME)

The following fields may be preset as they are not modified during OPEN (except for TC files): UFBERRAD, UFBEODAD, UFBRECAREA, UFBKEYAREA. For TC files, UFBRECAREA contains the address of the TC connection parameters.

For DMS/TX files, opened in I/O or Shared mode, initiates transaction processing on the file. See the VS DMS/TX Reference for more information.

Disk Considerations

For an existing disk file, if UFBFORG or UFBRECSIZE are supplied, they must agree with the values in the file's FDR.

UFBBUFSIZE may be supplied as the maximum buffer size required.

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) cannot be set by the user before OPEN; OPEN uses the binary value as a percentage. A value of 01-100 is accepted as the percentage value for packing. Any value outside this range is ignored; the default used produces full packing. UFBPKD and UFBPKI should not be modified after OPEN.

For Output mode disk files, UFBNRECS is to be supplied to OPEN as the basis for allocating file space on disk. OPEN saves the UFNRECS value in UFBOUTRECS and set UFBNRECS equal to zero.

Tape Considerations

For an existing tape file, if UFBFORG or UFBRECSIZE are supplied, they must agree with the values in the file's tape label.

Either the file name must be supplied or UFBTSEQ must be nonzero.

The user must specify a nonzero value for tape density in the case of 7-track tape; OPEN issues a set density command to set the user-specified density.

OPEN SVC rejects (with an explanatory respecify message) attempts to open a file on a device reserved by another task.

Error Handling

The OPEN macro EXIT parameter sets the high-order byte of the top word of the stack. This byte indicates 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 the OPEN SVC issuing a GETPARM (RESPECIFY). If the exit is taken, UFBFS1 is set to X'39', UFBFS2 is set to a mask for the appropriate condition; and UFBPREVO is set. The OPEN SVC may be reissued after an OPEN exit has been taken.

Open Exit Conditions

The open exit conditions for this macro are as follows:

- UFBFS2XFILE (X'80') -- Return if the file is not found (non-OUTPUT mode) 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 if the tape label type or tape density is not acceptable to the program.

- UFBFS2XPOS (X'04') -- Return for possession conflict. Possession conflict includes file already open by current program, file opened by other program and open modes conflict, and volume possession is exclusive for another user. Error further described in UFBXCODE.

- 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.

For the UFBXCODE, the values are described as follows:

- X'00' = No additional information
- X'01' = Device in use.
- X'02' = Device detached.
- X'03' = Volume exclusion.
- X'04' = File possession conflict
- X'05' = Paging file - system only.
- X'06' = Image file
- X'07' = Already open - this user.
- X'08' = Already open - this user.
- X'10' = Program requires 7-track tape while drive is 9-track or vice versa.
- X'11' = UFB FORG = PRINT while FDR FORG not equal to PRINT.
- X'12' = UFB FORG = PROG while FDR FORG not equal to PROG.
- X'13' = UFB FORG = CONSEC while FDR FORG not equal to CONSEC.
- X'14' = UFB FORG = WP while FDR FORG not equal to WP.
- X'15' = UFB FORG = INDEXED while FDR FORG not equal to INDEXED.
- X'16' = UFB FORG neither CONSEC nor INDEXED - error.

A NO-MESSAGE option is also available such that control returns 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.)

The following UFB fields are updated by OPEN:

- 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). The buffers marked CONTENTS NOT VALID (in UCBBCBFLAGS) and BUFFER IN PROTECTED MEMORY are also updated when required.

- The file status bytes (UFBFS1, UFBFS2) are set to '00'; UFBLF is set to 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)

- For output mode, UFBF4RLSE is set. This setting can be overridden by the user when the OPEN-GETPARM is issued or the bit can be cleared by the program after OPEN. The bit causes unused space in the file to be released by CLOSE.

- Program-supplied file, volume, and device information (as supplied through the OPEN parameter sections of the UFB) remains in the UFB, or is replaced with information acquired by OPEN through GETPARM.

Parameter Definitions

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.

MODE Specifies a value to be placed in the UFB to designate an open mode. This is done before the OPEN SVC is issued. This parameter is optional.

OUTPUT Mode that specifies that the file is opened for writing.

INPUT Mode that specifies that the file is opened for reading.

IO Mode that specifies that the file is opened for reading and updating.

EXTEND Mode that specifies that the file is opened for additions only.

SHARED Mode that specifies that the file is opened for use by more than one user. Opening a file in Shared mode also specifies that it is opened for reading and updating (I/O mode).

NOGETPARM Causes a type RD GETPARM to be issued rather than a type I, suppressing user interaction and causing procedure-supplied parameters to be ignored. This option should be used only when run time parameters have already been obtained, as 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 type ID GETPARM to be issued rather than a type I, suppressing user interaction as long as the values supplied in the UFB or through a procedure are syntactically correct. Even with the NOGETPARM or NODISPLAY options, a user interaction occurs if a field is semantically in error (e.g., an invalid device type).

EXIT A value that indicates 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 in this chapter 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 is created when the file is opened. Valid only for word processing files. This parameter 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 NO.

Stack On Input

One word on top of the stack, as follows:

```
                                        Lower
           |                       |    Address
           |0     1    2    3      |
  0(SP) |  |______________________ |
           |      |                |
           | (1)  | (2) Address    |
           |      |     of UFB     |
           |______|_______________ |
           |     Preceding         |    Higher
           |    Stack Data         |    Address
```

(1) EXIT condition mask - optional value specifying OPEN EXIT condition.

(2) Bytes 1 to 3 - address of the user file block (UFB) for the file to be opened.

Stack On Output

```
                                        Lower
                                        Address
           |                       |
           |______________________ |
  0(SP) |  |     Preceding         |    Higher
           |    Stack Data         |    Address
```

Example

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

4.2.52 PCEXIT - Modify Program Exception Exit Status (SVC 31)

Syntax

Format 1:

```
[label] PCEXIT  SET,{(list)},ADDRESS={(register)}
                    { ALL  }         { address   }
```

Format 2:

```
[label] PCEXIT  RESET
```

Format 3:

```
[label] PCEXIT  CANCEL
```

Function

Allows the user program to execute a user-written exception handling routine for user-selected program exceptions. This program exception handling status may also be reset or canceled.

When a program issues a LINK SVC, any current user program exception exit is eliminated, but the current status is preserved for restoration by UNLINK.

Parameter Definitions

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 link level (since a LINK from another program or Command Processor program initiation). All such status is discarded.

ADDRESS — Specifies the address of the entry point into a user supplied exception handling routine to which program control is transferred. A register specification in parentheses signifies that the register contains the exit address. An expression not in parentheses is evaluated as the exit address directly. This and the (list) parameter may be specified only with the SET parameter.

ALL — Specifies that all types of program exception interrupts are to be intercepted.

(list) — Where list may contain any of the following program exceptions 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).

Stack On Input

```
                                           Lower
               |                       |   Address
               |0     1      2      3  |
     0(SP)  |  |      | (2)  Address   |
               | (1)  | of Exit Routine|
               |      |  or unused     |
     4(SP)     |                       |
               | (3)  Exception Mask   |   Higher
               |      or unused        |   Address
               |      Preceding        |
               |     Stack Data        |
```

(1) Option flag:
  0 = Set new exit status (SET)
  1 = Remove most recent exit status (RESET)
  2 = Remove all exit status to most recent LINK (CANCEL)

(2) Exit address (SET option)

(3) Exception mask (SET option)

Stack On Output

```
                                                Lower
                 |                       |      Address
                 |_______________________|
      0(SP) |    |       Preceding       |      Higher
                 |      Stack Data       |      Address
```

Examples

```
LAB2      PCEXIT   SET,(OP,EX),ADDRESS=(R2)
+LAB2     PUSHC    0(4,0),*+10
+         B        *+8
+         DC       BL4'01010000000000000000000000000000'
+         PUSH     0,R2
+         SVC      31  (PCEXIT)

          PCEXIT   RESET
+         PUSHN    0,4
+         MVI      0(15),1
+         SVC      31  (PCEXIT)

          PCEXIT   CANCEL
+         PUSHN    0,4
+         MVI      0(15),2
+         SVC      31  (PCEXIT)
```

4.2.53 PROTECT - Protect a Disk File (SVC 42)

Syntax

Format 1:

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

Format 2:

```
[label] PROTECT {LIBRARY         },LIBRARY={address },
                {FILE={address }}         {'string'}
                      {'string'}

                VOLUME={address }[,OWNER={address }]
                       {'string'}        {'string'}

                [,FILECLAS={address }]
                           {'string'}

                [,RETPD=address][,EXPRDT=address]

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

Function

To update the protection information (protection class, owner of record, 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.

Restrictions

If the PLIST option is not utilized, PROTECT dynamically builds its parameter list on the stack, and it becomes the invoking program's responsibility to pop 32 bytes off the stack beyond the return code word on the stack.

The PROTECT SVC updates the protection information (protection class, owner ID, expiration date) for a disk file or library on a volume The structure of the volume table of contents is not affected by the change. The file must not be in use (open) when the protect is attempted or the SVC fails with a return code of 32.

To protect a file or library on a volume set, the root volume must be mounted. If not mounted, a generic mount will be issued for that volume (VSID).

Parameter Definitions

PLIST The address of a user-generated parameter list as used and described by the PROTECT SVC. If PLIST is specified, no other parameter may be specified.

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

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 specified library are to be modified. Use of this parameter is mutually exclusive with the FILE parameter.

FILE Specifies the name of the file whose protection attributes are to be modified. This parameter can be specified as a character string in single quotes which is the name of the file, or as an address expression pointing to an 8-byte field containing the file name. Use of this parameter is mutually exclusive with the LIBRARY parameter described above.

LIBRARY Specifies the name of the library or the name of the library storing the file whose protection attributes are to be modified. This parameter can be specified as a character string in single quotes which is the name of the library, or as an address expression that points to an 8-byte field containing the library name. This parameter is required if PLIST is not specified.

VOLUME Specifies the name of the volume containing the file or library whose protection attributes are to be modified. This parameter can be specified as a character string in single quotes which is the volume name, or as an address expression pointing to a 6-byte field containing the volume name. This parameter is required if PLIST is not specified.

OWNER If specified, indicates that the 3-byte owner of record protection attribute is to be modified. May be specified as an address expression that points to a 3-byte field which contains the owner of record, or as a character string in single quotes.

FILECLAS — If specified, indicates that the 1-byte file class protection attribute is to be modified. This parameter can be specified as an address expression that points to a 1-byte field which contains the new value, or as a character string in single quotes which is the new value.

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, format 00DDD+) is located. This parameter can be specified as a character string delimited by single quotes, in which case a constant is assumed. Use of this parameter is mutually exclusive with use of the EXPRDT parameter.

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, format YYDDD+) is located. This parameter can be specified as a character string delimited by single quotes, in which case a constant is assumed. Use of this parameter is mutually exclusive with use of the RETPD parameter.

RESTRICT — Specifies whether or not to ignore any current special access rights which may have been granted to the invoking program. If current special access rights ignored, the program is restricted to the user's logon access rights in determining whether the user may protect the specified file(s). Valid values are YES or NO. The default is NO.

Stack On Input

```
                                    Lower
              |                  |  Address
              |__________________|
     0(SP) |  |                  |
              | (1) Address of   |  Higher
              |__ Argument List__|  Address
              |    Preceding     |
              |    Stack Data    |
```

(1) Address of a parameter list constructed as follows:

```
|                    |
|   ARGUMENT LIST    |
| (2) Library name   | 8 bytes    Lower
| (3) File name-2    | 8 bytes    Address
|     or blank       |
| (4) Volume name    | 6 bytes
| (5) Option flag    | 1 byte
| (6) New protection | 1 byte
|     class          |
| (7) New owner ID   | 3 bytes
| (8) New expiration | 3 bytes
|     or retention   |            Higher
                                  Address
```

(2) File name 1

(3) File name 2

(4) Volume name

(5) Option flag: A 1- byte flag constructed as follows:

```
Bit 0   Must be 0
Bit 1   1 = Protect a library
Bit 2   1 = Limit access rights to USER LOGON rights
Bit 3   Must be 0
Bit 4   1 = Retention date supplied
Bit 5   1 = Expiration date supplied
Bit 6   1 = Set protection class
Bit 7   1 = set owner ID
```

(6) New protection class

(7) New owner ID

(8) New expiration date and retention period

Stack On Output

```
                                    Lower
         |                      |   Address
         |______________________|
0(SP) |  |                      |
         |     Return Code      |   Higher
         |______________________|   Address
         |     Preceding        |
         |    Stack Data        |
```

Output

Return codes in binary in the topword of the stack indicate the result of the request

Return Codes

Code	Description
0	The protection status for the specified file or library was successfully modified.
4	The indicated volume is not currently mounted.
8	The specified volume is currently being exclusively used by another user.
12	Insufficient stack space for buffers to process the RENAME request.
16	The specified library was not found.
20	The specified file was not found.
24	The user lacks update access for one or more of the files.
28	Unused.
32	The specified file is currently in use.
36	A VTOC error was encountered during processing - FDX1 and FDX2 do not agree.
40	A VTOC error was encountered during processing - FDX2 and FDR do not agree.
44	The address presented for the parameter list is invalid.
48	An I/O error occurred during processing - the VTOC is unreliable.
52	Open or protected files bypassed in protecting library.
56	Invalid new protection data.
60	Cluster communication failed.

Example

```
LAB1    PROTECT FILE=PROFILE,LIBRARY=PROLIBR,VOLUME=PROVOLUME,
                  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)
```

4.2.54 PUTPARM - Supply Program Parameters (SVC 33)

Syntax

```
[label] PUTPARM {DISPLAY},PRNAME='string',FMTLIST={(register)},
                {ENTER  }                         {  address }

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

                [{,PFKEY={ENTER }   }  ]
                         {number}

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

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 can not use PUTPARM to supply parameters for 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 buffer area in the program's modifiable data 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 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.

NOTE

Both the PUTPARM macro and the LINKPARM macro call the PUTPARM SVC (SVC 33). The PUTPARM macro supplys parameters only to another program, 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 LINKPARM has additional capabilities. The PUTPARM macro is kept solely for compatibility with existing programs.

The PUTPARM SVC has three major functions:

- PUT supplies parameters to another program's GETPARMs before issuing the LINK SVC to invoke the other program.

- CLEANUP cleans up the various internal data structures created by the PUT function.

- REFER allows the calling program access to any parameters which the user may have changed at GETPARM time, or to return the address of a previously created and labelled FMTLIST.

Both the PUTPARM macro and the LINKPARM macro call the PUTPARM SVC. The PUTPARM macro supplies parameters only to another program (the PUT function), while the LINKPARM macro accesses all the functions of the PUTPARM SVC.

The parameters to be supplied to the GETPARM are contained in a data structure, 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 buffer in the program's modifiable data area 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 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 are copied to the GETPARM KEYLIST (left-aligned and truncated). To solicit modifications by the user, A GETPARM workstation interaction may be requested by selecting the DISPLAY option; otherwise, a workstation interaction 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 are 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 for later use through the use of the LABEL parameter. This 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 is stored, causing GETPARM to reuse the labeled FMTLIST.

As an example of the backward reference facility, suppose that the program receiving parameters requests the same set of parameters several times and that the calling program is suppressing the workstation interactions. 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 interaction. 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 interaction.

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 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
```

Then, it 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
```

Then, it 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 PF key. This may be done in one of two ways, via either the PFKEY or the AID parameter. Both can pass the full range of 32 PF keys plus ENTER. PFKEY takes either the actual key number (1-32) or the keyword ENTER. AID takes the AID character of the PF key, where A-P correspond to PF keys 1-16 respectively, a-p correspond to PF keys 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 PF key is passed to GETPARM depends on whether the LINKPARM is a normal or a backward reference. In the normal case, the PF key is placed into the first byte of the FMTLIST addressed by FMTLIST by the LINKPARM macro. The original FMTLIST is modified. In the case of a backward reference, the PF key is placed onto the stack and then into the FMTLIST buffer. The original FMTLIST is not modified in this case.

Parameter Definitions

PUT — PUTPARM's primary use is to enable a program to supply parameters to a GETPARM issued by another program. 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.

CLEANUP — The CLEANUP option deallocates all the PRBs (and their associated FMTLISTs) chained to the program file block (PFB) of the current link level and link levels above. This option enables the user to free the buffers allocated for PUTPARM use. If no REFERLABEL is provided on the call, all PRBs and FMTLISTs at the current link level and link levels 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 use several LINKPARMs to prevent FMTLIST buffers from becoming full.

REFER,NOMERGE — The REFER,NOMERGE function of the PUTPARM SVC returns the address 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 that keep track of any GETPARM parameters that 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 that are present in the destination but not the source are left unchanged. Fields that 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.

DISPLAY — Requests GETPARM to display the screen for a workstation transaction.

ENTER — Requests GETPARM to bypass the screen display. A workstation transaction is suppressed.

PRNAME — The prname of the FMTLIST. Specified as a character string in single quotes, up to eight bytes in length.

FMTLIST The address of the FMTLIST in the format specified for the GETPARM SVC.

LABEL The label of the parameter reference block (PRB) to be used by the GETPARM SVC. Specified as a name of up to eight alphanumeric characters enclosed in single quotes.

REFERLABEL A name of up to eight alphanumeric characters that 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. REFERLABEL can be specified as an expression that addresses an 8-byte field containing the name of the FMTLIST, as a register in parentheses that points to an 8-byte field which contains the name of the FMTLIST, or as a character string in single quotes which is the name of the FMTLIST. 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.

AID The AID (Attention ID) character of a PF key to be passed to the GETPARM. AID characters are A-P (PF keys 1-16 respectively), a-p (PF keys 17-32 respectively), and @ (the ENTER key). AID can be specified as an expression that adresses a 1-byte field which contains the AID character, as a register in parentheses that points to a 1-byte field which contains 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 PF key 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.

Stack On Input

```
                |                  |   Lower
                |0    1    2   3   |   Address
     0(SP)  |   |_________________ |
                |    |     |       |
                | (1)| (2) | (3)   |
                |____|_____|_______|
     4(SP)      |                  |
                | (4) Address FMTLIST |
                |__________________|
     8(SP)      |                  |
                |                  |
                |__________________|
    12(SP)      |                  |
                | (5) prname       |
                |__________________|
    16(SP)      |                  |
                |                  |
                |__________________|
    20(SP)      |                  |
                | (6) LABEL        |   Higher
                |__________________|   Address
    24(SP)      |                  |
                |                  |
                |__________________|
                |    Preceding     |
                |    Stack Data    |
```

(1) Flag byte:
    Bit 0   1 = DISPLAY type
    Bit 1   1 = Search for a BWR FMTLIST
    Bit 2   1 = Clean up all PRBs and their FMTLISTs
    Bit 3   1 = Merge into user FMTLIST
    Bit 4   1 = Use repeat count
    Bit 5   1 = Cleanup BWRed PRB and FMTLIST only

(2) AID character for GETPARM

(3) Repeat count:
    X'00' = Never repeat
    X'01'-X'7FFF' = Repeat count
    X'8000' = Repeat indefinitely

(4) Address of a FMTLIST or backward referenced LABEL.

The FMTLIST to be constructed is as follows:

```
                 |                               |
                 |_______________________________|
       +0        | (a) PF Key     |(b) Number    |
                 |     Field      |    of fields |
       +4        |_______________________________|
                 | (c) Field Format              |
                 |     Control Block  1          |
                 |_______________________________|
                 |                               |
     +4 + BL     | (c) Field Format              |
            1    |     Control Block  2          |
                 |_______________________________|
                 |             .                 |
                               .
                 |             .                 |
                 |             .                 |
                               .
      n-1        |_____________._________________|
  +4 +    BL     |                               |
            i    | (c) Field Format              |
      i=1        |     Control Block N           |
                 |                               |
                 |_______________________________|
                 |                               |
                 |                               |
```

where BL = length of format control block

(a) A 1-byte receiving field for the corresponding AID character of the program function key received in a user response to a request for selection. This field may be set by a procedure specification of a function key number.

(b) A 1-byte binary count - number of field format control blocks.

(c) Format control block (variable length field) -- There are two formats for the format control blocks: one for control of the keyword/receiving field pairs, and the other to control the use of embedded text to be displayed. This field is repeated for each field to be displayed in the order they are to be displayed on the workstation screen.

Keyword/Receiving Field Field Format Control Block Structure

```
|   DATA STRUCTURE    |
|0    1    2    3     |
|     |    |    |     |    Lower
| (1) | (2)| (3)| (4) |    address
|_____|____|____|_____|
|                     |
| (5)  Keyword        |
|_____________________|
|                     |
| (6)  Receiving      |
|______Field__________|    Higher
|                     |    address
```

(1) Line-advance-count for display control. A 1-byte binary field.

(2) Space-advance-count for display control. A 1-byte binary field. Line advance takes place before space advance. Both take place before display of keyword and receiving field.

(3) Field error flag and receiving field entry restriction indicator - 1-byte binary field.

Bit 0: Field error flag

1 = Error - set by program to draw attention to fields in error. Reset by GETPARM.

Bits 5-7: Receiving field 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. An all blank response is 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 are ignored. An all blank response is treated as a legitimate NULL response. Field length is restricted to 16 characters.

4 = Uppercase alphanumeric - all entered letters are converted to uppercase. Legal nonblank response must be left-justified and consist entirely of the numerals 0-9, the letters A-Z, the special characters (@, #, or$), and trailing blanks. An all blank response is treated as a legal NULL response indicator. Maximum usable field length is 68 characters.

5 = Uppercase hexadecimal - all entered letters are converted to uppercase. 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. An all blank response is 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 are converted to uppercase. Legal nonblank responses are left-justified, beginning with a letter from A-Z, or one of the special characters (@, #, or $), and consist entirely of the numerals 0-9, the letters A-Z, the special characters, and trailing blanks. All blank responses are treated as a legal NULL response indicator. Maximum usable field length is 68 characters.

(4) A 1-byte binary receiving field length minus 1 (in characters).

(5) An 8-character, left-justified keyword used for display purposes (and to support noninteractive access via the procedure interpreter).

(6) A variable-length receiving field with default or current value in place.

Embedded Text Field Format Control Block Structure

```
|    DATA STRUCTURE     |
|0    1     2     3_____|
|     |     |     |     |   Lower
| (1) | (2) | (3) | (4) |   address
|_____|_____|_____|_____|
|                       |
| (5) Text              |
|                       |
|_______________________|   Higher
|                       |   address
```

(1) Line-advance-count for display control. A 1-byte binary field.

(2) Space-advance-count for display control. A 1-byte binary field. Line advance takes place before space advance. Both take place before display of keyword and receiving field.

(3) The value "-1" (= 255).

(4) Text field character length minus one. A 1-byte binary field.

(5) Character string to be displayed. Variable length field.

Stack On Output

```
                  |                      |   Lower
                  |______________________|   Address
       0(SP) |    |                      |
                  |      Return Code     |
                  |______________________|
       4(SP)      |                      |
                  |     FMTLIST Address  |   Higher
                  |_____of this PRB______|   Address
                  |     Preceding        |
                  |    Stack Data        |
```

Return Codes

Code	Description
0	Success.
8	Bad FMTLIST supplied to this SVC.
12	Error detected in previously constructed parameter reference blocks.

Examples

```
THERE     PUTPARM DISPLAY,PRNAME='ALIST',FMTLIST=(R2),LABEL='PBLOCK'
+THERE    DS     0H                       Place holder for label
+         PUSHC  0(8),=CL8'PBLOCK'        PRB Label
+         PUSHC  0(8),=CL8'ALIST'         PRNAME
+         PUSHA  0,0                      Unused
+         PUSHA  0,0(R2)                  FMTLIST
+         PUSHA  0,0                      Initial Flag bits
+         OI     0(15),X'80'              Display Flag
+         SVC    33                       (PUTPARM)


 HERE     PUTPARM ENTER,PRNAME='ABCDE',FMTLIST=ADDR1,LABEL='XYZ'
+HERE     DS     0H                       Place holder for label
+         PUSHC  0(8),=CL8'XYZ'           PRB Label
+         PUSHC  0(8),=CL8'ABCDE'         PRNAME
+         PUSHA  0,0                      Unused
+         PUSHA  0,ADDR1                  FMTLIST
+         PUSHA  0,0                      Initial Flag bits
+         SVC    33                       (PUTPARM)
```

4.2.55 READ - Read a Record

Syntax

```
[label] READ    [HOLD        ],UFB={(register)}[,COND={      integer      }]
             [REL         ]      { address  }       {absolute expression}
             [KEYED       ]                         {        15         }
             [NODATA      ]
             [TABS        ]
             [MOD         ]
             [ALTERED     ]
             [CONNECTPARM]
             [STATUS      ]
```

Function

Reads 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 parameter. Valid first parameters 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,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 Input, IO, or Shared mode, 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 record number within the file, or line numbers on the screen (from 1) to be read, is taken from the word addressed by UFBKEYAREA, and extends for the number of bytes specified by UFBKEYSIZE.

NOTE

Register 1 is loaded with the address of the UFB.

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
- 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 UFBERRAD is zero also, these conditions and I/O errors cause program terminations.

Parameter 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. REL is the default value 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 extends for the number of bytes specified in UFBKEYSIZE. The user's program should not modify UFBKEYSIZE.

HOLD Indicates that the record from a disk file may be rewritten by REWRITE or deleted by DELETE. Must be specified in order to successfully complete a REWRITE or DELETE 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.

NODATA Indicates that the record requested is to be read from the file in the manner indicated by other subparameters (including the HOLD subparameter), 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 telecommunications line connection parameters are to be read.

STATUS Indicates that telecommunications 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 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 tabs 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 remain 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 the 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 parameter 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)
+LAB     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
```

4.2.56 READFDR - Read File Descriptor Record (SVC 24)

Syntax

Format 1:

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

Format 2:

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

                VOLUME={(register)},AREA={(register)}
                       { 'string' }      { address   }
                       { address   }

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

                [,ALTVOL={(register)}][,VSID={    n     }]
                         { 'string' }        {(register)}
                         { address   }

                [,PLOG={NO  },PAREA={(register)}][,FDR3={    1     }]
                       {YES }       { address   }        {    n     }
                       {ONLY}                            {   BOTH   }
                                                         {(register)}

                [,FSN={    n     }]
                      {(register)}
```

Function

Allows user programs to locate a disk file on the specified volume and copy its file descriptor record(s) (FDRs) into the memory location denoted by the AREA parameter. Also, READFDR allows the caller to read a file prologue (only supported for word processing files) and to return the file prologue in a specified area.

If PLIST is not specified, then all parameters except FDR, FDR3, ALTLIB, and ALTVOL are required.

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

For files that are members of volume sets, a third file descriptor record (FDR3) exists which may be read by specifying the FDR3 parameter.

Parameter Descriptions

PLIST — A user-generated parameter list to be used by the READFDR SVC and in the form described in the SVC description. PLIST may be specified as a register in parentheses that points to the parameter list, or as an expression that addresses the parameter list. If this parameter is specified, then all other parameters are ignored.

FILE — The name of the file whose file descriptor records (FDRs) are to be accessed. It can be specified as a register in parentheses that points to the file name, a character string in single quotes which is the file name, or an expression that addresses a character string whose value is the file name.

LIBRARY — The name of the primary library to be searched for the file in question. It may be specified as in FILE above.

VOLUME — The name of the volume on which the primary library resides. It may be specified as in FILE above.

AREA — A user receiving area for storing the obtained file descriptor record(s). This must be 80 bytes if one FDR is requested and 160 bytes if FDR=BOTH or FDR3=BOTH is specified. It may be specified as a register in parentheses that points to the address of the receiving area, or as an expression that addresses a 4-byte field which contains the address of the receiving area.

FDR — This parameter indicates which FDR(s) to access. Valid values and their meanings are as follows:

- 1 -- Read the FDR1 only.
- n -- Read the (n-1)th FDR2 only, where n is an integer greater than 1. For example, 3 indicates that the second FDR2 is to be read.
- BOTH Read both the FDR1 and the first FDR2.

This parameter is optional and, if omitted, defaults to 1. (read FDR1 only).

ALTLIB — The name of a library to be searched if the file in question cannot be located in the primary library specified by the LIBRARY parameter. It may be specified as FILE above. This parameter is optional. However, if specified, then ALTVOL must also be specified.

ALTVOL
The name of the volume on which the alternate search library resides. Specified as FILE above. This parameter is valid only in conjunction with ALTLIB.

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

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

PAREA
Indicates the address of the receiving area for the file prologue. This parameter can be specified as a register in parentheses that contains the address of the receiving area, or as an address expression that points to a 4-byte area whcih contains the address of the receiving area.

VSID
Volume set identification number from 1 to 255. This parameter is only required when reading FDR2s for files contained in volume sets. For the root volume this value is one.

FDR3
This parameter specifies to read the FDR3 record for a file that is a member of a volume set. Valid values and their meanings are as follows:

- 1 -- Read the first FDR3 record for the file. (default value)
- n -- Read the nth FDR3 record (n >= 1).
- BOTH -- Read the FDR1 and the first FDR3.

FSN
Specifies the file sequence number to be read. A file sequence number represents a logically contiguous part of a file residing on a volume comprising one or more extents. When a file extends to a second (or third, etc.) volume, this number increments by one and represents the extents on the disk for the particular portion of the file. This value may range from 1 to 65536 and only applies to files that are members of volume sets and is required only when reading FDR2 records.

Stack On Input

```
                                                   Lower
               |                      |            Address
               |______________________|
      0(SP) |  |                      |
               | (1) Address of       |
               |_____Parameter List___|
      4(SP)    |                      |
               | (2) Reserved on      |            Higher
               |_______input__________|            Address
               |       Preceding      |
               |      Stack Data      |
```

(1) A pointer to a parameter list. The parameter list is constructed as follows:

(2) Reserved, must be 0.

```
                 |  PARAMETER  LIST   |
   PLIST |       |_(3)_Library_Name___| 8 bytes   Lower
                 |_(4)_File_Name______| 8 bytes   Address
                 |_(5)_Volume_Name____| 6 bytes
                 |_(6)_Option_Flag____| 1 byte
                 |_(7)_FDRN_Number____| 1 bytes
                 |_(8)_Memory_Address_| 4 bytes
                 | (9) Memory Address | 4 bytes
                 |_____2______________|
                 | (10) Volume set    | 4 bytes
                 |       information  |
                 |_______or Reserved__|
                 | (11) Alternate     | 8 bytes
                 |_______Library Name_|
                 | (12) Alternate     | 6 bytes  Higher
                 |_Volume Serial Nmbr_|          Address
                 |                    |
```

(2) Reserved on input.

(3) Primary search library name

(4) File name

(5) Volume name of primary search library

(6) Option flag:

Bits 0 to 1 - Reserved.

Bit 2 1 = Read FDR1 and first FDR3 into the 160 bytes of user supplied area. For volume set. FDRN value is ignored.

Bit 3 0 = Read FDR2 for multivolume files.
1 = Read FDR3 only for multivolume files.

Bit 4 1 = Alternate search library supplied. Last two words of the parameter list contain the alternate library and volume names.

Bit 5 1 = Read FDR1 and first FDR2 into the 160 bytes of user supplied area. FDRN value is ignored.

Bit 6 1 = Read file prologue along with any FDR record (valid only for word processing files).

Bit 7 1 = Read file prologue only (valid only for word processing files)

(7) FDRN number.

(8) Memory address to store requested information.

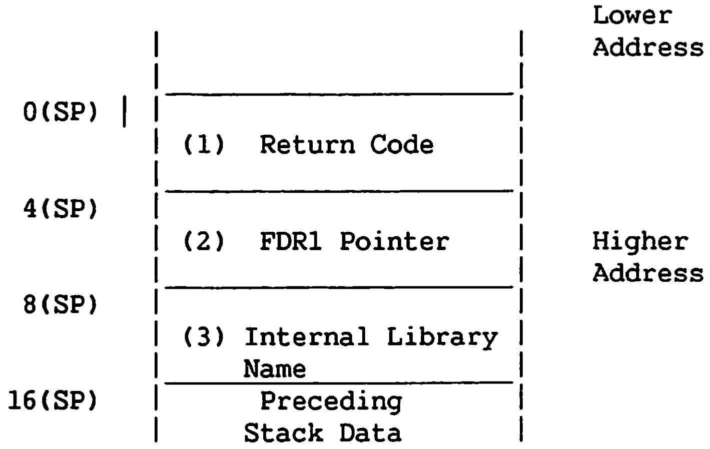
(9) Memory address to store additional requested information.

(10) For volume sets:
VSID (1 byte) volume set identification number (1-255)
FSN (2 bytes)
Unused (1 byte)
For single volumes:
Reserved - must be 0.

(11) Alternate search library name.

(12) Alternate search volume name.

Stack On Output



(1) Return code - If the function was not successful, the content of the second word on the stack is undefined.

(2) FDR1 pointer - If the READFDR is successful, contains the FDR address in the following format:
Byte 0 -- Record on block, from 0.
Bytes 1 to 3 -- Block on volume, from 0.

(3) Internal library name - 8 bytes, for Read FDR3=BOTH option. Otherwise, not present. For system use. 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 library.

Output

READFDR issues a return code to the user program in the stack top word indicating the success or failure of the operation, and the disk address of the FDR1 in the next stack word.

If return code = 0 (successful operation), the next word on the stack contains the disk address of the FDR record read, in the following format:

- Byte 0 -- Record on block, from 0.
- Bytes 1 to 3 -- Block on volume, from 0.

If the return code is not zero, then the contents of the next word on the stack are undefined.

Return Codes

Code	Description
0	File label copied into memory.
4	Volume not mounted.
8	Volume exclusively used by another user, no read.
10	Library not found.
12	All buffers in use, no read.
16	Library not found.
20	File label not found.
24	Attempt to read a file prologue when none was present.
28	Unused.
32	VTOC error - FDX1 and FDX2 do not agree.
36	VTOC error - FDX2 and FDR do not agree.
40	Invalid input parameters.
44	Disk I/O error - VTOC unreliable.
48	Read FDR2 but VSID, FSN not supplied.
52	Read FDR1 & FDR3 in single volume.
56	Unused.
60	GETHEAP failed.
64	Cluster communication failed.

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
 LAB1     READFDR FILE=(R1),LIBRARY='SYSLIB',VOLUME=SYSVOL,                -
                AREA=MYAREA,FDR=BOTH,ALTLIB='SYSLIB2',ALTVOL=SYSVOL
+LAB1     PUSHN 0,50                 GET SPACE ON STACK FOR PLIST       01\
+         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 IGNORED FOR FDR=BOTH)
+         MVC   24(4,15),MYAREA      SET FDR RECEIVING AREA ADDRESS
+         XC    32(4,15),32(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


 LAB3     READFDR FILE=MYFILE,LIBRARY=(R1),VOLUME=SYSVOL,AREA=(R6)
+LAB3     PUSHN 0,36                 GET SPACE ON STACK FOR PLIST       01\
+         MVC   0(8,15),0(R1)        SET LIBRARY NAME
+         MVC   8(8,15),MYFILE       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    32(4,15),32(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
```

```
 LAB4     READFDR FILE=MYFILE,LIBRARY=SYSLIB,VOLUME=SYSVOL,AREA=MYAREA, -
                FDR3=BOTH,VSID=2
+LAB4     PUSHN 0,44                GET SPACE ON STACK FOR PLIST     01\
+         MVC   0(8,15),SYSLIB      SET LIBRARY NAME
+         MVC   8(8,15),MYFILE      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
+         MVC   24(4,15),MYAREA     SET FDR RECEIVING AREA ADDRESS
+         OI    22(15),X'10'        SET READ FDR3 FLAG ON            01\
+         MVI   22(15),X'40'      SET FLAG TO READ FDR1 AND 1ST FDR3 01\
+         XC    32(4,15),32(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
 SYSVOL   DC C'SYSTEM'
 SYSLIB   DC C'@SYSTEM@'
 MYAREA   DS 80F
 MYFILE   DC C'TAXMAN'
```

4.2.57 READVTOC - Read Volume Table of Contents (SVC 19)

Syntax

```
[label] READVTOC    OPTION={LIBRARIES }[,PLIST={(register)}]
                           {ATTRIBUTES}        { address  }
                           {EXTENTS   }
                           {FILES     }
                           {BLOCKS    }

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

                [,COUNT={(register)}][,START={(register)}]
                        { integer  }         { address  }
                                             {    1     }

                [,OFB={(register)}][,VSID={(register)}]
                      { address  }        { address  }
                                          {    0     }
```

Function

Provides information from a disk volume table of contents (VTOC). Specific functions are described under OPTION.

To read information from the VTOC of a specified volume. Five options can be performed:

- Read VTOC attributes: extents in use, number of unused blocks in the VTOC, total number of directories on the volume, total number of files on the volume, total number of free extents on the volume, total size of free extents, and the largest free extent.

- List the free extents on the volume starting from a specified extent.

- List the directories and the corresponding number of files on each volume starting from a specified directory in the VTOC.

- List files in a specified directory starting from a specified file in the directory.

- Read consecutive control blocks in the VTOC starting from a specified block and place them in the file pointed to by the OFB pointer.

Restrictions

The area addressed by PLIST must be in the user's modifiable data area. If any parameters are supplied as character strings (and in some other cases), the user must allow for generation of a literal pool.

Parameter Descriptions

OPTION One of the following options, coded as shown, that indicates the type of information is desired. This parameter 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 on a single volume or the root volume of a volume set.

FILES Lists m (m=COUNT) file names starting from nth (n=START) file in specified library on a single volume or the root volume of a volume set.

BLOCKS Reads consecutive VTOC blocks starting from the block specified by the START parameter for the number of blocks specified by the COUNT parameter into the file specified by the OFB parameter.

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 parameters (in this case, it is assumed that the user has placed values in the PLIST for parameters that would otherwise have been required).

VOLUME An expression, a register in parentheses that points to a 6-byte name, or a literal in single quotes that indicates the volume from which VTOC information is desired. Required for all options (unless PLIST is specified).

LIBRARY An expression, a register in parentheses that points to an 8-byte name, or a literal in single quotes that indicates the library about which VTOC information is desired. Required when OPTION=FILES (unless PLIST is specified). Not valid for non-root volumes of volume sets.

COUNT A number or a register in parentheses that contains a number which indicates how many items (see OPTION description) are requested. Required for all options (unless PLIST is specified).

START An expression, or a register in parentheses that contains a number which indicates which item (see OPTION description) is the first item requested. Required for all OPTIONs (unless PLIST is specified). START=1 is the default. If PLIST is specified and the default START value is not desired, START= must also be coded (see examples).

OFB The address or a register in parentheses that contains the address of the open file block. The file specified must be opened for output with enough space allocated to accommodate m VTOC blocks (as specified in BLOCKS).

VSID Volume set identification number from 0 to 255. Must be supplied for volumes that are members of a volume set. Ignored for single volumes.

Stack On Input

```
                                      Lower
            |                      |  Address
            |______________________|
    0(SP) | |                      |
            | (1)  Address of      |  Higher
            |______Parameter List__|  Address
            |      Preceding       |
            |      Stack Data      |
```

(1) The address of a parameter list. The parameter list is constructed as follows:

```
                  |                       |
PLIST ADDR |      | (1) Volume name       |  6 bytes   Lower
                  |                       |            Address
                  | (2) Option number     |  1 byte
                  |                       |
                  | (3) VSID              |  1 byte
                  |                       |
                  | (4) Number of items   |  2 bytes
                  |                       |
                  | (5) Starting item     |  2 bytes
                  |     number            |
                  | (6) Library name or   |  8 bytes
                  |     OFB pointer       |
                  | (7) Variable length   |
                  |     memory space for  |
                  |     output informa-   |             Higher
                  |     tion              |             Address
                  |  Preceding data       |
                  |                       |
```

(1) Volume name -- bytes 0-5

(2) Option number -- byte 6

0 = Read VTOC attributes.

(a) VTOC extents in use; number of unused blocks in VTOC. If the number of unused blocks is greater than or equal to 255, then 255 is returned.

(b) Total number of directories on volume; total number of files on volume.

(c) Total number of free extents on volume; total size of free extents.

(d) m largest free extents on volume.

1 = List M free extents on volume starting from the nth free extent.

2 = To list M directories and the corresponding number of files in each directory on the volume starting from the nth directory in the VTOC.

3 = To list M files in a specified directory starting from the Nth file in the directory.

4 = To read M consecutive control blocks in the VTOC starting from the Nth block in VTOC and put them in the file specified by the given OFB pointer.

(3) VSID -- byte 7 is the volume set identification number (0-255)

(4) Number of times (m >= 1) -- bytes 8-9

(5) Starting number (n >= 1) -- bytes 10-11

(6) Directory name -- bytes 12-19, or
OFB pointer -- bytes 12-15, or
not used

(7) Output area -- not used on input, bytes 20-X. The size depends on option specified in byte 6 and must be big enough to hold the desired output argument list.

Stack On Output

```
                                          Lower
              |                      |    address
              |______________________|
     0(SP) |  |                      |
              |     Return Code      |    Higher
              |______________________|    Address
              |       Preceding      |
              |      Stack Data      |
```

When the return code equals 0, the input argument list is replaced by one of the following output argument lists, depending on the option specified:

Option 0

ARGUMENT LIST		
Number of Unused Blocks In VTOC	1 byte	Lower Address
Number of VTOC Extents in Use	1 byte	
1st VTOC Extent Start and End Block Numbers	6 bytes	
2nd VTOC Extent Start and End Block Numbers	6 bytes	
3rd VTOC Extent Start and End Block Numbers	6 bytes	
Total Number of Directories on Volume	2 bytes	
Total Number of Files on Volume	2 bytes	
Total Number of Free Extents	2 bytes	
Total Size of Free Extents	4 bytes	
1st Largest Free Extent Start and End Block Numbers	6 bytes	
.		
.		
.		
mth Largest Free Extent Start and End Block Numbers	6 bytes	Higher Address

Option 1

ARGUMENT LIST		
Total Number of Free Extents on Volume	2 bytes	Lower Address
Total Number of Free Extents Listed	2 bytes	
nth Free Extent Start and End Block Numbers	6 bytes	
.		
.		
.		
(n+m-1)st Free Extent Start and End Block Numbers	6 bytes	Higher Address

Option 2

ARGUMENT LIST		
Total Number of Directories on Volume	2 bytes	Lower Address
Total Number of Directories Listed	2 bytes	
Directory Name n	8 bytes	
Number of Files in Directory N	2 bytes	
. . .		
Directory Name n+m-1	8 bytes	Higher Address
Number of Files in Directory Name n+m-1	2 bytes	

Option 3

ARGUMENT LIST		
Total Number of Files in Directory	2 bytes	Lower Address
Total Number of Files Listed	2 bytes	
Filename n	8 bytes	
. . .		
Filename n+m-1	8 bytes	Higher Address

Option 4

ARGUMENT LIST		
Total VTOC Size in Blocks	2 bytes	Lower Address
Number of Blocks Read	2 bytes	
Unchanged	16 bytes	Higher Address

Additional output for option number 4: nth through (n+m-1)st VTOC control blocks copied to the file specified by the given OFB.

Output

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 (previous contents of the register are lost).

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

A return code is issued in the word at the top of the stack.

Return Codes

Code	Description
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.
36	Option not allowed across cluster.
40	GETHEAP failed.
44	Cross cluster communication failed.
48	Files and libraries not available for non-root volumes.

Examples

```
 READ1  READVTOC OPTION=ATTRIBUTES,VOLUME='VOLVO',COUNT=32,START=(R8)
+READ1  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 ADDRESS TO STACK
+              SVC    19 (READVTOC)          ISSUE READVTOC SVC
```

```
READ2   READVTOC OPTION=EXTENTS,VOLUME=VSCBNAME,COUNT=(R7)
+READ2  DS    0H
+             LR    1,R7                    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   R7,8(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

READ3   READVTOC OPTION=LIBRARIES,VOLUME=(R6),COUNT=32
+READ3        DS    0H
+             LA    1,340                   SIZE OF PARAMETER LIST
+             PUSHN 0,0(,1)                 SPACE FOR PARAMETER LIST
+             MVC   8(2,15),=Y(32)          SET COUNT FIELD
+             MVI   6(15),2                 INSERT OPTION BYTE
+             MVC   10(2,15),=Y(1)          SET START FIELD
+             MVC   0(6,15),0(R6)           MOVE IN VOLUME NAME
+             PUSH  0,15                    PARAMETER ADDRESS TO STACK
+             SVC   19 (READVTOC)           ISSUE READVTOC SVC


READ4   READVTOC OPTION=FILES,VOLUME='SYSTEM',LIBRARY='SYSS',                   -
                   COUNT=16
+READ4        DS    0H
+             LA    1,148                   SIZE OF PARAMETER LIST
+             PUSHN 0,0(,1)                 SPACE FOR PARAMETER LIST
+             MVC   8(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 ADDRESS TO STACK
+             SVC   19 (READVTOC)           ISSUE READVTOC SVC


READ5   READVTOC OPTION=BLOCKS,VOLUME=VSCBNAME,COUNT=(R3),                          -
                   START=(R4),OFB=(ROFB)
+READ5        DS    0H
+             LA    1,20                    SIZE OF PARAMETER LIST
+             PUSHN 0,0(,1)                 SPACE FOR PARAMETER LIST
+             STH   R3,8(15)                SET COUNT FIELD
+             MVI   6(15),4                 INSERT OPTION BYTE
+             STH   R4,10(15)               SET START FIELD
+             MVC   0(6,15),VCBSER          MOVE IN VOLUME NAME
+             ST    ROFB,12(,15)            SET OFB ADDRESS
+             PUSH  0,15                    PARAMETER ADDRESS TO STACK
+             SVC   19 (READVTOC)           ISSUE READVTOC SVC
```

```
 READ6  READVTOC PLIST=(RLIST),START=(R3)
+READ6  DS    0H
+             STH   R3,10(RLIST)            SET START FIELD
+             PUSH  0,RLIST                 PARAMETER ADDRESS TO STACK
+             SVC   19 (READVTOC)           ISSUE READVTOC SVC

          READVTOC OPTION=EXTENTS,START=,PLIST=(R2)
+         DS    0H
+         MVI   6(R2),1             . INSERT OPTION BYTE .
+         PUSH  0,R2                   . PARAMETER ADDRESS TO STACK .
+         SVC   19 (READVTOC)          . ISSUE READVTOC SVC .
          END BEGIN
```

4.2.58 RECEIVE - Receive Telecommunications I/O (SVC 3)

Syntax

```
[label] RECEIVE DATA,OFB={ address },RECAREA={ address },
                        {(register)}         {(register)}

              LENGTH={     address      }[,IOCWOPTS={ address }]
                     {    (register)    }           {(register)}
                     {self-defining term}

              [,COMMAND={      address     }]
                        {    (register)    }
                        {self-defining term}
```

Function

Initiates a data reception operation between the operating system and the data link processor (DLP). The RECEIVE macroinstruction invokes the XIO SVC (SVC 3) to perform the physical I/O operation. XIO SVC checks that the specified communication channel is opened, and that the communication channel and the DLP are not reserved by another task. A CHECK for completion of the I/O operation is not implicit, and must be affected by waiting for reception of the I/O status word (IOSW) using the TCIO option of the CHECK facility. See the XIO macro for further information.

Parameter Definitions

OFB — A required parameter that defines the address of the open file block (OFB) for the VS-DLP I/O channel to be used in the I/O operation. The OFB address can be obtained from the interprocessor control block (IPCB). The OFB address for the VS/DLP I/O channel device is stored in the IPCB by the IPOPEN SVC when the communication channel is opened. The OFB parameter can be specified as an address expression that points to a 4-byte field which contains the OFB address in its low-order three bytes, or as a register in parentheses that contains the address of the OFB in the low-order three bytes.

COMMAND — A required parameter which enables the user to supply a value for the command byte (byte 0) of the I/O Command Word (IOCW) constructed by the XIO SVC. The command byte defaults to X'40' (the READ command) for the RECEIVE DATA macro. COMMAND can be specified as an address expression that points to a 1-byte field which contains the command byte, as a register in parentheses that contains the command byte in its low-order byte, or as a self-defining term.

RECAREA — A required parameter that defines the address of the reception area for the receipt of the input from the READ operation. The RECAREA parameter is the value placed by the XIO SVC in the data address field of the IOCW (bytes 1-3). This parameter can be specified as an address expression, or as a register in parentheses that contains the address of the reception area in the low-order three bytes.

LENGTH — A required parameter that defines the maximum length of the input from the READ operation. The LENGTH parameter is the value placed by the XIO SVC in the data count field of the IOCW (bytes 4-5). This parameter can be specified as an address expression that points to a 2-byte area which contains (in binary) the length in bytes, as a register in parentheses that contains the length in bytes in its low-order two bytes, or as a self-defining term.

IOCWOPTS — Values for the last three bytes (bytes 6-8) of the 9-byte I/O command word (IOCW) can be supplied with the IOCWOPTS parameter. The last three bytes default to zeroes. This parameter can be specified as an address expression that points to a 3-byte field which contains the option bytes, or as a register in parentheses that contains the three option bytes in its low-order three bytes.

Output

High-order halfword of return code field contains residual block counts:

- Return codes 4, 8 -- Specified block size minus number of bytes actually read or written.

- Other return codes -- 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 other return codes are set, the operation has been suppressed.

A return code is issued by the XIO SVC in the stack top word. The low-order halfword of the return code field contains binary return codes.

Return Codes

Code	Description
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 be either in the 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 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.

Example

```
 GETDATA RECEIVE DATA,OFB=F1OFB,COMMAND=CMDBYTE,RECAREA=INBUF,          -
                LENGTH=BUFLNGTH
+GETDATA PUSHA  0,0                   CLEAR IOCW OPTIONS AREA
+        PUSHA  0,0                   CLEAR NEXT 4 BYTES OF SPACE
+        MVC    0(2,15),BUFLNGTH      SET DATA LENGTH
+        PUSHA  0,INBUF               SET DATA TRANSFER ADDRESS
+        MVC    0(1,15),CMDBYTE       SET IOCW COMMAND CODE BYTE
+        PUSHA  0,0                   SPACE FOR OFB ADDRESS
+        MVC    0(4,15),F1OFB         PUSH ADDRESS OF THE "OFB"
+        MVI    0(15),X'01'           MARK AS 'TC XIO'
+        SVC    3  (XIO)
```

4.2.59 REGS - Register Equation

Syntax

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

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 that make use of system macroinstructions. Register names are as follows:

General		Floating Point	
Register Numbers	Names	Register Numbers	Names
0	R0	0	F0
1	R1,AP	2	F2
2	R2	4	F4
3	R3	6	F6
4	R4	5	R5
6	R6	7	R7
8	R8	9	R9
10	R10	10	R11
12	R12	13	R13,EP
14	R14	15	R15,SP

Parameter Definitions

FP   If NO is specified, symbolic names for the floating-point registers are not generated. The default is YES.

Example

```
              REGS
+R0      EQU 0
+R1      EQU 1
+AP      EQU 1
+R2      EQU 2
+R3      EQU 3
+R4      EQU 4
+R5      EQU 5
+R6      EQU 6
+R7      EQU 7
+R8      EQU 8
+R9      EQU 9
+R10     EQU 10
+R11     EQU 11
+R12     EQU 12
+R13     EQU 13
+EP      EQU 13
+R14     EQU 14
+R15     EQU 15
+SP      EQU 15
+F0      EQU 0
+F2      EQU 2
+F4      EQU 4
+F6      EQU 6
```

4.2.60 RENAME - Rename a Disk File (SVC 26)

Syntax

Format 1:

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

Format 2:

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

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

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

Format 3:

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

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

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

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

Function

To rename a disk file or a library on a volume. A full RENAME (renaming both library and file) may alter the volume table of contents (VTOC); otherwise, the structure of the VTOC is not altered. Unless the OPEN=YES option is specified, no file that is to be renamed may be open when the RENAME is attempted. A full RENAME is equivalent to moving a file from one library to another on the same volume.

Restrictions

If any of the parameters are 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 undesired 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 required a minimum of 9K of stack for buffer space to rename both a library and a file (full RENAME).

Parameter Definitions

PLIST The address of a user-generated parameter list to be used by the RENAME SVC as described in the RENAME SVC. If PLIST is specified, no other parameter may be specified.

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

If PLIST is not specified, the macro generates code to dynamically build a parameter list on the stack prior to issuance of the RENAME SVC.

NOTE

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 bytes if full RENAME) off the stack beyond the return code word.

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

FILE Specifies the name of the file to be renamed. This parameter can be specified as a character string in single quotes that is the name of the file, or as an address expression that points to an 8-byte field which contains the file name. Use of this parameter is mutually exclusive with the LIBRARY parameter 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 parameter may be specified as a character string in single quotes that is the name of the library, or as an address expression that points to an 8-byte field which contains the library name.

VOLUME Specifies the name of the volume that contains the file or the library to be renamed. This parameter may be specified as a character string in single quotes that is the volume name, or as an address expression that points to a 6-byte field which contains the volume name. This parameter is required if PLIST is not specified.

NEWNAME Specifies the new name of the file or library being renamed. This parameter can be specified as a character string in single quotes that is the new file name or library name, or as an address expression that points to an 8-byte field which contains the new file name or library name. This parameter is required if PLIST is not specified.

NEWLIB The name of the library in which the renamed file is to be placed. If omitted, then the same library as specified by the LIBRARY parameter is assumed. This parameter can be specified as a character string in single quotes that is the new file name or library name, or as an address expression that points to an 8-byte field which contains the new file name or library name.

RESTRICT Specifies whether the RENAME SVC is to ignore any special access rights that may have been granted to the invoking program. If special access rights are ignored, program is restricted to the user's logon access rights in determining whether the user can RENAME the specified file(s). Valid values are YES or NO; the default is NO.

BYPASS Specifies whether the RENAME SVC is to bypass checking the expiration date of the file(s) being renamed. Valid values are YES or NO; the default is NO.

Stack On Input

```
          |                      |   Lower
          |______________________|   Address
  0(SP) | |                      |
          |  (1)  Address of     |   Higher
          |______Parameter List_ |   Address
          |      Preceding       |
          |     Stack Data       |
```

(1) Address of an argument list constructed as follows:

```
|    PARAMETER LIST    |
| (2)  Old lib name    | 8 bytes    Lower
| (3)  Old file name   | 8 bytes    Address
| (4)  New file name   | 8 bytes
| (5)  Volume name     | 6 bytes
| (6)  Option flag     | 1 byte
| (7)  Not used        | 1 byte
| (8)  New library     | 8 bytes    Higher
|      name            |            Address
|______________________|
|                      |
```

(5) Option flag byte:

Bit 0   1 = Bypass file expiration date check.
Bit 1   1 = Access rights limited to user LOGON rights.
Bit 2   1 = Allow rename when open for exclusive I/O by the caller.
Bit 3   1 = Rename both file and library.
Bit 4   Reserved, must be 0.
Bit 5   Reserved, must be 0.
Bit 6   Reserved, must be 0.
Bit 7   Reserved, must be 0.

Stack On Output

```
                                          Lower
           |                        |     Address
           |________________________|
   0(SP) | |                        |
           |      Return Code       |     Higher
           |________________________|     Address
           |       Preceding        |
           |       Stack Data       |
```

Return Codes

Code   Description

0   The specified file or library was successfully renamed.

4   The indicated volume is not currently mounted.

8   The specified volume is currently being exclusively used by another user.

12   Insufficient stack space for buffers to process the RENAME request.

16   The specified library was not found.

20   The specified file was not found.

Code	Description
24	The user lacks update access for one or more of the files to be renamed. No files were renamed.
28	One or more specified files were not past expiration date. No files were renamed.
32	The specified file is currently in use and no rename occurred.
36	A VTOC error was encountered during processing. FDX1 and FDX2 do not agree.
40	A VTOC error was encountered during processing. FDX2 and FDR do not agree.
44	The address presented for the parameter list is invalid.
48	An I/O error occurred during processing. The VTOC is unreliable.
52	The new file name or library name already exists.
56	The new filename is invalid or a number sign (#) is the first character.
60	The VTOC is currently full. Insufficient space exists for the new FDX1/FDX2 (full RENAME only).
64	The reserved bits (bits 5-7) in the parameter list options byte are nonzero.
68, 72, 76	Unused.
80	Cluster communication failed.

Example

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

4.2.61 RESETIME - Remove Timer Interval (SVC 32)

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.

Stack On Input

```
                                                Lower
                |                       |       Address
                |0     1     2     3    |
       (SP) |   |_______________________|
                |     |                 |
                | (1) | (2)Interval or  |       Higher
                |_____|____Time of day__|       Address
                |       Preceding       |
                |       Stack Data      |
```

```
(1)  Type of request - byte 0
        High bit:    0 = Set interval
                     1 = Reset interval
        Next bit:    0 = Interval supplied
                     1 = Time of day supplied
```

(2) Interval or time of day - bytes 1-3. In hundredths of a second. (Ignored for RESET.) A time value less than the present time results in immediate expiration. When referring to a time earlier than the current time (in order to refer to the next day), specify the time plus 24 hours.

Stack On Output

```
                                                Lower
                                                Address
                |                       |
                |_______________________|
      0(SP) |   |       Preceding       |       Higher
                |       Stack Data      |       Address
```

Example

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

4.2.62 RETURN - Return to Invoker

Syntax

```
[label] RETURN  [UNLINK][,CODE={(register)}][,COND={integer}]
                              { address }        {   15  }
                              {    0    }
```

Function

The RETURN macroinstruction is used to (conditionally) exit from a program to the system where 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 resulted in entry to the program or subprogram. The contents of general registers 1-14 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. (RETURN CODE=(0) leaves register 0 unchanged.)

Restrictions

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

Parameter Definitions

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 parameter.

CODE — If the CODE parameter is supplied, register 0 is loaded with the number specified, or from the register specified. In this case, the following instruction is generated:

```
LA 0,number   or   LR 0,Rn
```

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 parameter is specified.

Example

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

4.2.63 REWRITE - Rewrite a Record

Syntax

```
[label] REWRITE [{TABS    },]UFB={(register)}[,COND={integer}]
                {SELECTED}      { address  }       {  15    }
                {  REL    }
```

Function

Rewrite 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. 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 released by an intervening operation on any other shared file). 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.

NOTE

Register 1 is loaded with the address of the UFB.

An error condition discovered on REWRITE results 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:

- 00 -- Normal return, success
- 23 -- Return to UFBEODAD, block beyond end of file for block-level I/O or invalid key (REWRITE REL).
- 30 -- Return to UFBERRAD, permanent I/O error.
- 34 -- Return to UFBERRAD, order check on workstation.
- 95 -- Return to UFBERRAD, invalid function or function sequence (includes key validation failure for indexed file REWRITE).

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

Parameter Descriptions

TABS Indicates that bytes 5-14 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.

REL Rewrites a record for a relative file. A READ HOLD operation is not required before performing the rewrite. For relative files only.

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 parameter 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
```

4.2.64 ROLLBACK - DMS/TX Transactions Rollback (SVC 76)

Syntax

```
ROLLBACK    RETCODE={(register)}[,CANCEL={NO }][,ACK={NO }]
                    {  address }         {YES}       {YES}

            [,{LEVELS={(Register)}}]
                      {  address }
             {ALL={YES}           }
                  {NO }
```

Function

The ROLLBACK macro provides a means for undoing a DMS/TX transaction or subtransaction. Rolls back the current DMS/TX transaction which restores all updated records to their previous values. Applies to all databases in use by this task. See the VS DMS/TX Reference for more information.

Parameter Definitions

ACK — YES specifies that an acknowledge GETPARM is to be issued for errors. NO specifies that no acknowledge GETPARM is to be issued for errors. NO is the default.

CANCEL — YES specifies that the update is to be cancelled when an error is detected. NO is the default.

RETCODE — The address at which to store the return code.

ALL — YES specifies rollback all levels. If NO is specified and LEVEL is not specified, one level will be rolled back.

LEVELS — Identifies the number of levels to rollback. This parameter can be specified as either a register specification or the address of a storage location that contains the number of levels.

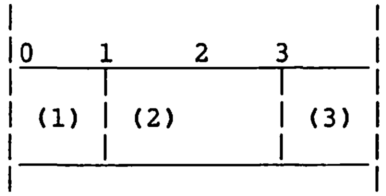
Input To The SVC

Register 1 points to a 2-word argument list constructed as follows:

```
           |                      |
           |______________________|
  (R1) |   |  (1)                 |     Lower
           |  Address of          |     Address
           |  Return Code_________|
           |  (2)                 |
           |  Address of          |     Higher
           |  Function Code_______|     Address
           |                      |
```

(1) Address of a fullword structure that indicates where to store the return code from the ROLLBACK SVC.

(2) Address of a 1-word structure that contains the error handling code and function code constructed as follows:



(1) Error handling code, byte 0:
Bit 0 1 = Cancel.
Bit 1 1 = Issue acknowledge GETPARM on error.
Bit 2 Reserved, must be 0.
Bit 3 Reserved, must be 0.
Bit 4 Reserved, must be 0.
Bit 5 Reserved, must be 0.
Bit 6 Reserved, must be 0.
Bit 7 Issue database name message header.
(2) Reserved, must be 0 (bytes 1,2)
(3) Function code, byte 3
Bit 1 1 = Rollback transaction.

Output

The return code is stored in the address supplied on input to the SVC.

Return Codes

Code	Description
0	Success.
4	No recovered files are open.
8	DMX/TX not supported on this system.
12	Invalid function request.
16	Invalid parameter or parameter list.
20	Unable to process before image journal for this task. Run DMSTX utility on this database.
24	Error encountered on this file during rollback. Run DMSTX utility on this file.

Code	Description
28	Specified mark not found. The entire transaction has been rolled back.
32	Unable to set file crash status. File may contain uncommitted updates.
36	Unable to set database crash status. Database may contain uncommitted updates.
40	Error freeing locks.
44	Bad nesting (returned by VSETX).

Example

```
         ROLLBACK RETCODE=(R2),CANCEL=YES,ACK=YES
+        PUSHA 0,=A(-2147483647)
+        PUSH  0,R2                return code
+        LR    1,15
+        SVC   76 (ROLLBACK)
+        POPN  0,2*4
         ROLLBACK RETCODE=(R2),CANCEL=YES
+        PUSHA 0,=A(-2147483647)
+        PUSH  0,R2                return code
+        LR    1,15
+        SVC   76 (ROLLBACK)
+        POPN  0,2*4
```

4.2.65 SCRATCH - Scratch a File (SVC 27)

Syntax

Format 1:

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

Format 2:

```
[label] SCRATCH {LIBRARY         },LIBRARY={address },
                {FILE={address   }         {'string'}
                      {'string'  }

                VOLUME={ address}[,RESTRICT={NO }]
                       {'string'}           {YES}

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

Function

Deletes a disk file or a library of disk files from a volume, making the space utilized by the file(s) available for reallocation. Removes 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.

When deleting files from volume sets, the system issues a mount request for all volumes which have extents for the file. If SCRATCH terminates abnormally, any volume which has not been mounted up to this time will have lost space. That is, the disk space allocated to the file will not be returned to the volume's list of free extents. Lost space cannot be retrieved. Compress-in-place (CIP) cannot retrieve it.

Restrictions

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.

Parameter Definitions

PLIST The address of a SCRATCH parameter list. If PLIST is specified, no other parameter 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 specified library are to be deleted. Use of this parameter is mutually exclusive with the FILE parameter.

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

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

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

RESTRICT If NO is specified, or the parameter is omitted, the SCRATCH operation proceeds to utilize 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 parameter 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.

Output

A return code is issued in the top word of the stack. 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 the three high-order bytes are zeroed.

When the last file in a library is deleted, the library is eliminated.

Stack On Input

```
                                         Lower
               |                      |  Address
               |______________________|
     0(SP) |   |                      |
               |  (1)  Address of     |  Higher
               |_______Argument List__|  Address
               |      Preceding       |
               |     Stack Data       |
```

(1) The address of an argument list constructed as follows:

```
         |    ARGUMENT LIST    |
         | (2)  Library Name   | 8 bytes    Lower
         | (3)  File Name      | 8 bytes    Address
         | (4)  Volume Name    | 6 bytes
         | (5)  Option Flag    | 1 byte
         | (6)  Not used       | 1 byte     Higher
         |                     |            Address
```

(5) Option flag:

```
   Bit 0   1 = Bypass expiration date check.
   Bit 1   1 = Delete all closed and expired files in
   library for which update access is allowed.  Delete
   library if all files are closed and expired.
   Bit 2   1 = File access rights limited to user LOGON
           rights.
   Bit 3   Reserved.
   Bit 4   Reserved.
   Bit 5   Reserved.
   Bit 6   Reserved.
   Bit 7   Reserved.
```

Stack On Output

```
                                          Lower
            |                      |      Address
            |0    1     2     3    |
    0(SP) | |                 |    |
            |  (1) Lost       | (2)|      Higher
            |      Extents    |    |      Address
            |      Preceding       |
            |     Stack Data       |
```

(1) Total size of lost extents in blocks during scratch. No extent lost if size equals 0.

(2) Return code.

Return Codes

Code	Description
0	File or library successfully deleted.
4	Volume not mounted.
8	Volume used exclusively by other user.
12	All buffers in use, no deletion.
16	Library not found.
20	File not found.
24	Update access denied, no deletion (single-file deletion only).
28	Unexpired file, no deletion (single-file deletion only).
32	File in use, no deletion.
36	VTOC error, FDX1 and FDX2 do not agree.
40	VTOC error, FDX2 and FDR do not agree.
44	Invalid argument list address.
48	I/O error, VTOC unreliable.
52	Open, protected, or unexpired file bypassed in library being deleted.
56	Link down or unable to allocate resources. Remote file or library not deleted.
60	Success, but same volumes skipped.

Example

```
 SCRFILE  DC CL8'MYFILE'
 SCRLIBR  DC CL8'OURLIBRY'
 SCRVOL   DC CL6'SYSTEM'
          SCRATCH FILE=SCRFILE,LIBRARY=SCRLIBR,VOLUME=SCRVOL
+         PUSHN 0,8
+         MVI   7(15),0
+         MVI   6(15),B'00000000'
+         MVC   0(6,15),SCRVOL      VOLUME
+         PUSHC 0(8),SCRFILE        FILE
+         PUSHC 0(8),SCRLIBR        LIBRARY
+         PUSH  0,15
+         SVC   27                  (SCRATCH)


          SCRATCH FILE='MYFILE',LIBRARY='MYLIB',VOLUME='MYVOL'
+         PUSHN 0,8
+         MVI   7(15),0
+         MVI   6(15),B'00000000'
+         MVC   0(6,15),=CL6'MYVOL' VOLUME
+         PUSHC 0(8),=CL8'MYFILE'   FILE
+         PUSHC 0(8),=CL8'MYLIB'    LIBRARY
+         PUSH  0,15
+         SVC   27                  (SCRATCH)
```

4.2.66 SET - Set Task-Related Defaults (SVC 35)

Syntax

```
[label] SET [FILECLASS={(register)}]        [,FORM#={(register)}]
                       { 'string' }                 { 'string' }
                       {  address }                 {  address }

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

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

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

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

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

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

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

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

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

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

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

Function

Allows user programs to set default values for task related parameters according to the parameters specified. These values which are stored in a tasks' ETCB are used by the various system utilities and SVCs. None of the parameters have defaults and any unspecified parameters are unaffected.

Restrictions

All library and volume name specifications (except literals) must reference 8- and 6-byte fields respectively, as the SET SVC cannot determine the length of the character string and assumes the maximum.

Parameter Definitions

> NOTE
>
> All parameters are optional (although at least one should be specified).

Parameters can be specified as

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

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

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

FILECLASS Default file protection class. The following values are valid:

- # -- Accessible only by system security administrators and the owner-of-record.

- $ -- READ-only files. READ access granted to all users regardless of the individual's 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.

FORM# Default form number for print files. The association of a form number with a specified form is installation-defined. This 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.

INLIB Default input library name. This pair of parameters is used primarily by the OPEN SVC to locate files opened as input files.

INVOL Default input volume name.

JOBCLASS Default job class for a background job. Background jobs are processed according to the job class priority hierarchy specified from Workstation 0. 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 is specified, then the job has no time limit.

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 released for execution.

LINES Default number of lines-per-page. This parameter is used primarily by the print functions of system utilities. This number must be in the range 0 to 255.

OUTLIB Default output library name. This pair of parameters is used primarily by the OPEN SVC to assign files opened as output files.

OUTVOL Default output volume name.

PRINTER Default printer device number for on-line printing. This parameter 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 are as follows:

- O -- ONLINE, printing is done using the printer as a direct output device; a print file is not created.

- S -- SPOOL, print files are created and are queued by the system task (@SYSTSK@) for printing at the earliest opportunity.

- K -- KEEP, print files are created but are not queued for printing by the system task.

- H -- HOLD, print files are placed in the user's print library and are queued by the system task, but are not printed until requested by the system operator or the user.

PROGLIB Default program/procedure library name. This pair of parameters is used only in procedures, for programs run by those procedures. These parameters identify the library and volume that are to serve as the default user program library and volume for all programs run by a procedure.

PROGVOL Default program/procedure volume name.

PRTCLASS Default print class. This parameter determines the class to which print requests sent to the system task are 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.

PRTFILECLAS Default spool file class.

RUNLIB Default program/procedure execution library name. The RUNVOL and RUNLIB parameter pair are used by the command processor RUN command to locate programs and procedures to be executed.

RUNVOL Default program/procedure execution volume name.

SPOOLIB Default spool library name constructed from user ID or background task number.

SPOOLSYS Default system name for remote print routing.

SPOOLSYSRC Required if SPOOLSYS is specified. Contains the return code for setting SPOOLSYS with one of the following possible values:

- 0 = Successful
- 4 = System name not found
- 8 = GETMEM failure
- 12 = XMIT failure

SPOOLVOL Default volume for assignment of spooled (print) files.

WORKVOL Default volume for assignment of work files.

Initializes corresponding entries in the issuing task's extended task control block (ETCB) with values supplied by a parameter list placed on the stack.

Stack On Input

```
                                           Lower
                |                       |  Address
                |_______________________|
      0(SP)   | | Addr of PROGVOL  value|
      4(SP)     | Addr of PROGLIB  value|
      8(SP)     | Unused                |
     12(SP)     | Addr of INVOL    value|
     16(SP)     | Addr of INLIB    value|
     20(SP)     | Unused                |
     24(SP)     | Addr of OUTVOL   value|
     28(SP)     | Addr of OUTLIB   value|
     32(SP)     | Addr of SPOOLVOL value|
     36(SP)     | Addr of WORKVOL  value|
     40(SP)     | Addr of PRINTER  value|
     44(SP)     | Addr of PRNTMODE value|
     48(SP)     | Addr of FILECLAS value|
     52(SP)     | Addr of LINES    value|
     56(SP)     | Addr of PRTCLASS value|
     60(SP)     | Addr of FORM#    value|
     64(SP)     | Addr of RUNVOL   value|
     68(SP)     | Addr of RUNLIB   value|
     72(SP)     | Addr of JOBQUEUE value|
     76(SP)     | Addr of JOBCLASS value|
     80(SP)     | Addr of JOBLIMIT value|
     84(SP)     | Addr of SPOOLIB  value|
     88(SP)     | Addr of PRTFILECLAS   |
     92(SP)     | Addr of LOGBLKPTRvalue|
     96(SP)     | Addr of SPOOLSYS value|   Higher
    100(SP)     | Addr of SPOOLSYSRC    |   Address
                |      Preceding        |
                |     Stack Data        |
```

The parameter list contains addresses of the values used to initialize the ETCB symbols. A zero placed in the corresponding parameter list position indicates that the ETCB symbol is not to be initialized. The following list contains the procedure keyword for ETCB symbols that may be initialized and the expected length of the data:

Procedure Keyword	Length
PROGVOL	6
PROGLIB	8
SPARE	0
INVOL	4
INLIB	8
SPARE	0
OUTVOL	6
OUTLIB	8
SPOOLVOL	6
WORKVOL	6
PRINTER	1
PRNTMODE	1
FILECLAS	1
RESERVED	1
PRTCLASS	1
FORM#	1
RUNVOL	6
RUNLIB	8
JOBQUEUE	1
JOBCLASS	1
JOBLIMIT	4
SPOOLIB	4
PRTFILECLAS	1
LOGBLKPTR	4
SPOOLSYS	8
SPOOLSYSRC	4

Stack On Output

On completion, the SVC removes the parameter list from the stack.

```
                                              Lower
                                              Address
                 |                     |
                 |_____________________|
   0(SP) |       |      Preceding      |      Higher
                 |     Stack Data      |      Address
```

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 LITERAL75
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
```

4.2.67 SETIME - Set Interval Timer (SVC 32)

Syntax

Format 1:

```
[label] SETIME UNTIL={(register)}
                     { address  }
```

Format 2:

```
[label] SETIME CSEC={(register)}
                    {  address }
```

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 is set.

Parameter Definitions

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 results in immediate expiration.

CSEC — Either a register specification in parentheses, containing the number (in binary) of 1/100-second units to delay processing, or an expression, not in parentheses, for the required number of 1/100-second units. The value cannot exceed one day.

Stack On Input

```
                                       Lower
                |                |     Address
                |0   1   2   3   |
                |____________________
      (SP) |    |    |           |
                | (1)|(2)Interval or|  Higher
                |____|  Time of day|   Address
                |    Preceding   |
                |    Stack Data  |
(1)  Type of request - byte 0
        Bit 0:  0 = Set interval
                1 = Reset interval
        Bit 1:  0 = Interval supplied
                1 = Time of day supplied
```

(2) Interval or time of day 0 - bytes 1 to 3, in hundredths of a second. (Ignored for RESET.) A time value less than the present time results in immediate expiration. If requesting a time of day earlier than the current time (in order to refer to the next day), specify the time plus 24 hours.

<u>Stack On Output</u>

```
                                           Lower
                                           Address
               |                      |
               |______________________|
      0(SP) |  |      Preceding       |    Higher
               |     Stack Data       |    Address
```

<u>Example</u>

```
 LAB1     SETIME CSEC=55
+LAB1     PUSHA  0,55
+         MVI    0(15),0     UNITS
+         SVC    32 (SETIME)
```

4.2.68 SETRECOV - DMS/TX Set File Recovery Options (SVC 82)

Syntax

```
SETRECOV {ATTACH    },RETCODE={(register)},FILE={(register)},
         {DETACH    }         { address  }     { address  }
         {RESETCRASH}                          { 'string' }

                       VOLUME={(register)},LIBRARY={(register)},
                              { address  }         { address  }
                              { 'string' }         { 'string' }

                  [,DATABASE={(register)}][,CANCEL={NO }][,ACK={NO }]
                             { address  }          {YES}       {YES}
                             { 'string' }
```

Function

Attaches or detaches a file with recovery blocks to a DMS/TX database, or clears a crash status. The file must be an indexed file, with recovery blocks to which the user has update rights. The file must be closed at the time of the request. For the ATTACH function, the file must not be already attached to a database, and the database must exist. For the DETACH and RESET CRASH STATUS functions, the file must be attached to some database. See the VS DMS/TX Reference for more information.

Parameter Definitions

ATTACH Function request -- attach file to database.

DETACH Function request -- detach file from database.

RESETCRASH Function request -- clear crash status.

DATABASE A 6-character database name. Optional, only used with ATTACH.

VOLUME A 6-character field designating the volume name of the file to be attached or detached.

LIBRARY An 8-character field designating the library name of the file to be attached or detached.

FILE An 8-character field designating the file name to be attached or detached.

RETCODE The address at which to store a fullword binary return code indicating the success or failure of the function.

CANCEL — An optional parameter that specifies whether to cancel the transaction on error. The default is NO.

ACK — An optional parameter that specifies whether to produce a message when an error is encountered. The default in NO.

Input To The SVC

Register 1 points to an 8-word argument list that is constructed as follows:

```
          |______________________|
   (R1) | | (1)                  |   Lower
          | Address of the       |   Address
          |_Return Code__________|
          | (2)                  |
          | Address of the       |
          |_File Name____________|
          | (3)                  |
          | Address of the       |
          |_Library Name_________|
          | (4)                  |
          | Address of the       |
          |_Volume Name__________|
          | (5)                  |
          | Address of the       |
          |_Function Request_____|
          | (6)                  |
          | Address of the       |
          |_Error Option_________|
          | (7)                  |
          | Address of the       |
          |_RECOPTS______________|
          | (8)                  |
          | Address of the       |   Higher
          |_Database Name________|   Address
          |                      |
```

(1) Address of where to store the return code from SETRECOV.

(2) Address of an 8-character file name.

(3) Address of an 8-character library name.

(4) Address of a 6-character volume name.

(5) Address of the function request code which is one character that has one of the following values:
A - Attach
D - Detach
R - Reset crash status

(6) Address of the error option code which is a 1-character field that has one of the following values:
blank - No special handling
C - Cancel on error
A - Issue acknowledge GETPARM with return code

(7) Pointer to a 4-byte structure which must contain zeros.

(8) Address of a 6-character database name. (Attach function only)

Output From SVC

The return code from the SETRECOV SVC is stored in the address supplied on input to the SVC.

Output

A return code is issued in the topword of the stack.

Return Codes

Code	Description
0	Success.
4	DMS/TX not supported on this system.
8	Invalid recovery option value.
12	Invalid function request value.
16	Invalid parameter or parameter list.
20	Database option library @DMSTX@ not found on the IPL volume.
24	Database option file not found.
28	Unexpected READFDR error when trying to find database option file.
32	File has no recovery blocks allocated.
36	File is already attached to a database.
40	File is not attached to a database.
44	Recovery on consecutive files not supported.
48	File not properly closed, reorganization recommended.

Code	Description
52	Errors in alternate index structures probable, reorganization required.
56	File possession conflict.
60	Volume not mounted.
64	Library not found.
68	File not found.
72	User has insufficient access rights to file.
76	File contains uncommitted updates, must be recovered or reset before detach.
80	Unexpected OPEN error.
84	Unexpected UPDATFDR error.
88	Unexpected CLOSE error.
92	Unexpected UPDATLSB error.
96	Insufficient buffer space.
100	VTOC error on IPL volume.
104	I/O error encountered on file.
108	Spare bytes in recovery option must be zero.
112	Communication with Message Handler failed.
116	Cross-cluster communication failed.

Examples

```
         SETRECOV ATTACH,DATABASE=TRANS,VOLUME=(R2),LIBRARY=(R3),           -
               FILE='TODAY',RETCODE=(R5)
+        PUSHA 0,TRANS                Database
+        MVI   0(15),X'80'            SET 'LAST' PARAMETER BIT
+        PUSHA 0,=A(0)                File Recovery Option
+        PUSHA 0,=A(4)                PUSH FUNCTION PARAMETER
+        PUSH  0,R2                   Volume
+        PUSH  0,R3                   Library
+        PUSHA 0,=CL8'TODAY'          File
+        PUSH  0,R5                   return code
+        LR    1,15
+        SVC   82 (SETRECOV)
+        POPN  0,7*4


         SETRECOV DETACH,VOLUME=(R2),LIBRARY=(R3),FILE='YSTRDAY',           -
               RETCODE=(R5)
+        PUSHA 0,=A(8)                PUSH FUNCTION PARAMETER
+        MVI   0(15),X'80'            SET 'LAST' PARAMETER BIT
+        PUSH  0,R2                   Volume
+        PUSH  0,R3                   Library
+        PUSHA 0,=CL8'YSTRDAY'        File 02
+        PUSH  0,R5                   return code
+        LR    1,15
+        SVC   82 (SETRECOV)
+        POPN  0,5*4


         SETRECOV RESETCRASH,VOLUME=(R2),LIBRARY=(R3),FILE='TODAY',         -
               RETCODE=(R5)
+        PUSHA 0,=A(16)               PUSH FUNCTION PARAMETER
+        MVI   0(15),X'80'            SET 'LAST' PARAMETER BIT
+        PUSH  0,R2                   Volume
+        PUSH  0,R3                   Library
+        PUSHA 0,=CL8'TODAY'          File
+        PUSH  0,R5                   return code
+        LR    1,15
+        SVC   82 (SETRECOV)
+        POPN  0,5*4
```

4.2.69 START - Start File Processing in Specified Mode or at Specified Record Location

Syntax

```
[label] START   {  IO   },UFB=(register)[,COND={integer}]
                {OUTPUT }                      {address}
                {EXTEND }                      {   15  }
                { BEGIN }
                {  END  }
                {  SKIP }
                {  EQ   }
                {  GT   }
                {  GE   }
                {  LT   }
                {  LE   }
                { ATTNT }
                { WAIT  }
                { HOLD  }[,RANGE][,RETRIEVAL][,LIST]
                {RELEASE}
                {TCWAIT }[,MULTIPLE     ][,TIMEOUT={(register)}]
                        [(MULTIPLE,ATTN][          { address  }]
                {HALTIO }
```

Function

The function of START differs for the following various file types:

- Consecutive disk files (normal DMS) -- START IO, OUTPUT or EXTEND are valid for files opened in IO, OUTPUT or EXTEND modes and alter the current open mode. 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 then puts 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 then adds 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

  START END is valid in IO mode, whether opened in IO mode or subsequently started in IO mode. START END sets the end of file to the current position within the file, effectively deleting all records in the file past that point. For example, a READ of the Nth record in the file followed by a START END leaves N records in the file.

- Consecutive disk files (normal DMS) -- START BEGIN and START SKIP are valid in INPUT and IO modes. A READ NEXT issued after START BEGIN reads 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) skips over n records and reads the record after them (n greater than 0), merely reads the next record (n=0), rereads the current record (n=-1), or reads a preceding record (n -1).

- Consecutive disk and magnetic tape files (physical access method) -- START WAIT is valid in INPUT, OUTPUT, or IO modes. The program pauses 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)

- Indexed and relative 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 (similar to 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.

  - LT -- The first record with a key less than the supplied key is sought. For relative disk files only.

  - LE -- The first record with a key less than or equal to the supplied is sought. For relative disk files only.

  After a successful START function, a succeeding READ (without KEYED option) reads the record located by START. Successive READs 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 occurs. UFBGKSIZE is set to zero by every such START function.

Possible invalid-key and error conditions 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

- 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 descriptions in the VS Principles of Operation manual.

- 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.

- Telecommunications 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 completion 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 parameter 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 is used. In either case, TIMEOUT specifies the time interval in seconds.

Table 5-1 summarizes the uses of START.

Table 5-1. START - Modes of Use with Disk Files

	OPEN for Input	OPEN for Output	OPEN for I/O	OPEN for Extend	OPEN for Shared I/O

Consecutive RAM	SKIP
	BEGIN
		IO
		OUTPUT
		EXTEND
			SKIP
			END
			BEGIN
			IO
			OUTPUT
			EXTEND
				IO
				OUTPUT
				EXTEND
					SKIP
					END
					BEGIN

Indexed RAM	EQ GT GE		EQ GT GE		EQ GT GE HOLD RELEASE
BAM		IO OUTPUT EXTEND		IO OUTPUT EXTEND	
PAM	WAIT	WAIT IO OUTPUT	WAIT		
Relative RAM	EQ GT GE LT LE	IO OUTPUT EXTEND	IO OUTPUT EXTEND EQ GT GE LT LE	IO OUTPUT EXTEND	

Parameter Definitions

IO
OUTPUT
EXTEND
BEGIN
SKIP
END
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 parameter of the JSCI instruction by which the START function is entered. Thus the START is made conditional. COND=15 is the default. Register 11 is loaded with the UFB address even when the condition is not satisfied.

Example

```
 OUTPUT   START GE,UFB=(R2)
+OUTPUT   LR    1,R2                  SET REGISTER 1
+         MVI   16(1),X'03'           GREATER THAN OR EQUAL TO
+         JSCI  15,16(1)              START FUNCTION
```

4.2.70 START HOLD/RELEASE - Hold/Release Resource

Syntax

Format 1:

```
[label]     START  HOLD,{        RANGE          },
                       {       RETRIEVAL        }
                       {         LIST           }
                       {   (RANGE,RETRIEVAL)    }
                       {     (RANGE,LIST)       }
                       {   (RETRIEVAL,LIST)     }
                       {(RANGE,RETRIEVAL,LIST)}

                   UFB={(register)}
                       {expression}
```

Format 2:

```
[label]     START   RELEASE,UFB={(register)}
                                {expression}
```

Function

The START HOLD function requests holds on resources in a data file and also requests extension rights. The options are as follows:

- The RANGE option indicates that a range of records in a file is to be held.

- The RETRIEVAL option allows more than one user to hold the same resource for retrieval only. If this option is not specified, the default is hold for update; in this case only one user can hold the resource.

- The LIST option allows the user to set up a list of resources to be held, and later add to the list by issuing another START HOLD. The programmer indicates that the list is complete by issuing a START HOLD without the list option; the actual hold of all the resources in the list then takes place.

The START RELEASE function releases all held resources in the specified file.

Parameter Definitions

UFB A register in parentheses or an address expression pointing to the UFB of the data file whose records are being held.

Examples

```
 LAB1   START HOLD,(RANGE,RETRIEVAL,LIST),UFB=RSUFB
+LAB1   LA    1,RSUFB                      SET REGISTER 1
+       MVI   16(1),B'11010100'            OPTIONS
+       JSCI  15,16(1)                     START FUNCTION


 LAB2   START HOLD,UFB=(R1)
+LAB2   MVI   16(1),B'100000000'           OPTIONS
+       JSCI  15,16(1)                     START FUNCTION


 LAB1   START RELEASE,UFB=RSUFB
+LAB1   LA    1,RSUFB                      SET REGISTER 1
+       MVI   16(1),B'00100000'            OPTIONS
+       JSCI  15,16(1)                     START FUNCTION

 LAB2   START RELEASE,UFB=(R1)
+LAB2   MVI   16(1),B'00100000'            OPTIONS
+       JSCI  15,16(1)                     START FUNCTION
```

4.2.71 SUBMIT - Submit Job or Print Request (SVC 46)

Syntax

Format 1:

```
[label] SUBMIT  JOB[,PLIST={(register)}][,PROCNAME={(register)}]
                           { address  }            {'string'  }
                                                   { address  }

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

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

                   [,STATUS={'RUN'   }][,DISP={'REQUEUE'}]
                            {'HOLD'  }        { address }
                            { address}        {         }

                   [,CPULIMIT=({(register)}[,{'CANCEL'}])]
                               { address  }  {'PAUSE' }
                                             {'WARN'  }
                                             { address}

                   [,CPUSECONDS=({(register)}[,{'CANCEL'}])]
                                 { address  }  {'PAUSE' }
                                               {'WARN'  }
                                               { address}

                   [,DUMP={'YES'   }] [,PERMANENT={YES}]
                          {'NO'    }              {NO }
                          {'PROG'  }
                          {'string'}
```

Format 2:

```
[label] SUBMIT  PRINT[,PLIST={(register)}][,FILENAME={(register)}]
                             { address  }            { 'string' }
                                                     {  address }

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

                    [,PRTCLASS={(register)}][,FORM#={(register)}]
                               {'string'  }         { integer  }
                               { address  }         { address  }
```

```
[,COPIES={(register)}][,STATUS={'SPOOL' }]
         { integer  }         {'HOLD'  }
         { address  }         { address}

[,DISP={'REQUEUE'}]
       {'SAVE'   }
       { address }
```

Function

Dynamically requests the queuing of a print file for printing, a procedure file for execution, and the transmitting or retrieving of files from one computer system to another.

If the initial parameter is JOB, SUBMIT requests the queuing of a procedure file for execution as a noninteractive job.

If initial parameter is PRINT, SUBMIT requests the queuing of a print file for printing.

Parameter Definitions

PLIST — A 44-byte user-supplied parameter list (fullword aligned) for use by the SUBMIT SVC and constructed as shown in the Stack on Input section.

If PLIST is specified, then the remaining parameters are optional and, if present, are used to modify the parameter list in place. The default values of any omitted parameters 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 parameters are used to build a parameter list on the stack. The default values of omitted parameters 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 parameters 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 resides.

VOLUME — The name of the volume on which the procedure/file resides.

JOBNAME

An optional user-supplied name for the job to be submitted (limited to 8 characters).

JOBCLASS/
PRTCLASS

The class to which the job or print request is to be assigned. Valid values are the letters A-Z.

FORM#

The number of the form on which to print this file. This number must be in the range 0-254 (decimal).

COPIES

The number of copies of this file to be printed. This number must be in the range 1-32767 (decimal). The default value is 1.

CPULIMIT

The total amount of CPU time that this job may use is specified by the first parameter and the action to be taken if that limit is exceeded is specified by the second parameter.

The actual CPU time may be specified as a register in parentheses or an expression that addresses a 4-byte field which contains the limit in timer units. A value of 0 implies that the job has no limit and any action indicated by the second parameter will be ignored. The default is zero (no limit).

The action to be taken upon completion may be specified either as one of the following character strings in single quotes, or as an expression that addresses a 1-byte field which contains the appropriate flag value (see PLIST entry for byte 37 (JOB) above):

- 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.

The default is WARN. CPULIMIT can be specified without the action to take upon completion parameter. The action to take upon completion parameter can be specified without CPULIMIT only when PLIST is also specified.

CPUSECONDS

This parameter specifies, in seconds, the total amount of CPU time that a job can take. The second parameter specifies the action to be taken. See CPULIMIT for a description of the valid values for the second parameter. CPUSECONDS and CPULIMITS are mutually exclusive.

STATUS The initial status of the request when it is placed on the queue. It may be specified either as one of the following character strings in single quotes or as an expression that addresses a 1-byte field which contains the appropriate flag value (see PLIST entry for byte 36 (JOB) or byte 26 (PRINT) above):

- 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.

The default is RUN/SPOOL.

DISP The action to be taken at completion of the request. It may be specified as a character string in single quotes or as an expression that addresses a 1-byte field which contains 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 or re-printing (for PRINT requests, this implies SAVE).

SAVE Do not delete this file after printing (PRINT only).

DUMP The action to be taken in the event of an abnormal termination. It may be specified as one of the following character string in single quotes or as an expression that addresses a 1-byte field which contains the appropriate flag value (see PLIST entry for byte 31 above):

- 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 that is terminating abnormally.

The default is PROG.

PERMANENT For background jobs, if YES is specified, the system re-initiates the job when the system is re-IPLed. The initial SUBMIT parameter values will still be in effect. NO is the default.

Stack On Input

```
                                       Lower
              |                     |  Address
              |0    1    2     3    |
     0(SP)    |     |               |
              | (1) | (2) Address   |  Higher
              |     | Parameter List|  Address
              |     Preceding       |
              |    Stack Data       |
```

(1) Operation code - binary value from 0 - 255.

(2) Address of a parameter list constructed in the following manner for the particular type of request:

SUBMIT JOB

ARGUMENT LIST		Lower Address
(1) Procedure Name	8 bytes	
(2) Library Name	8 bytes	
(3) Volume Name	6 bytes	
(4) Job Name	8 bytes	
(5) Job Class	1 byte	
(6) Dump Options	1 byte	
(7) CPU Time Limit	4 bytes	
(8) Job Type	1 byte	
(9) Hold/Active	1 byte	
(10) Other Flags	1 byte	
(11) Reserved	5 bytes	Higher Address

(1) The name of the procedure (PROCNAME) to be run.

(2) The name of the library in which the procedure resides.

(3) The name of the volume on which the procedure resides.

(4) A user-supplied job name or spaces.

(5) The job class to which this job is to be queued.

(6) The action to be taken in case of an abnormal termination of this job:
X'C0' - Produce a dump for this job (DUMP=YES).
X'80' - Do not produce a dump for this job (DUMP=NO).
X'00' - Produce a dump only if requested by the program terminating abnormally (DUMP=PROG).

(7) The CPU time limit (in timer units) imposed upon this job. If zero, then the job has no time limit.

(8) Job type:
X'80' - Permanent
X'00' - Not permanent

(9) The initial status of this job when it is queued.
X'80' - STATUS=HOLD. Not eligible for scheduling until released by the operator or the submitter.
X'00' - STATUS=RUN. Eligible for scheduling upon submission of the request.

(10) 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 CANCEL nor PAUSE is specified and a CPU time limit has been set, then a warning is issued.)
X'04' - REQUEUE this job after execution.
X'01' - CPU limits are in seconds.

(11) Reserved, must be 0.

SUBMIT PRINT

ARGUMENT LIST		Lower Address
(1) Print File Name	8 bytes	
(2) Library Name	8 bytes	
(3) Volume Name	6 bytes	
(4) Print Class	1 byte	
(5) Form Number	1 byte	
(6) # of Copies	2 bytes	
(7) Hold/Active	1 byte	
(8) Options	1 byte	
(9) Reserved	16 bytes	Higher Address

(1) The name of the file (FILENAME) to be printed.

(2) The name of the library in which the file resides.

(3) The name of the volume on which the file resides.

(4) The print class (PRTCLASS) to which this file is to be queued.

(5) The form number (FORM#), in binary, of this file to be printed.

(6) The number of COPIES (in binary) of this file to be printed.

(7) The initial status of this file when it is queued.

(8) Options.
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.

(9) Whether or not this file should be requeued, saved, or scratched after printing:
X'40' - REQUEUE this file after printing.
X'20' - SAVE this file after printing.

(10) Reserved, should be 0.

SUBMIT TRANSMIT or SUBMIT RETRIEVE

A 224-byte data structure that is constructed as follows:

		Lower Address
(1) Local file information	71 bytes	
(2) Remote file information	71 bytes	
(3) File Name Format	1 byte	
(4) Remote File Type	1 byte	
(5) Location	16 bytes	
(6) Group	16 bytes	
(7) Replace	1 byte	
(8) Status	1 byte	
(9) Disposition	1 byte	
(10) Transfer Dispo	1 byte	
(11) Options	28 bytes	

(1) Local file information can be one of three different formats depending upon the type of file specified in (3) below.

(a) Format for VS file:

	Lower Address
(a) File Name	8 bytes
(b) Library	8 bytes
(c) Volume	6 bytes
(d) Reserved	49 bytes

(b) Format for word processing document:

	Lower Address
(a) Document ID	4 bytes
(b) Document Library	1 byte
(c) Password	6 bytes
(d) Reserved	5 bytes
(e) Document Volume	6 bytes
(f) Reserved	49 bytes

(c) Format for OIS file:

	Lower Address
(a) OIS File Name	71 bytes

(2) Remote file information can be any of the formats described below.

(a) Format for VS file:

	Lower Address
(a) File Name	8 bytes
(b) Library	8 bytes
(c) Volume	6 bytes
(d) Reserved	49 bytes

(b) Format for word processing document:

	Lower Address
(a) Document ID	4 bytes
(b) Document Library	1 byte
(c) Password	6 bytes
(d) Volume	8 bytes
(e) Reserved	52 bytes

(c) Format for OIS file:

	Lower Address
(a) OIS File Name	71 bytes

(3) File name format
X'00' - undefined
X'01' - VS file
X'02' - WP document file
X'03' - VS/OIS file
X'04' - OIS file

(4) Remote file type
X'00' - undefined
X'01' - VS file
X'02' - WP document file
X'03' - VS/OIS file
X'04' - OIS file

(5) Location

(6) Group

(7) Replace
X'00' - No
X'80' - Yes

(8) Status
X'00' - Active
X'80' - Hold

(9) Disposition
X'00' - Save
X'80' - Delete

(10) Transfer disposition
X'00' - Store
X'01' - Print
X'02' - Run

(11) Options specifies print files or run options for procedure files. One of the following two formats must be supplied:

(a) Print options

(12) Print Class	1 byte
(13) Form #	1 byte
(14) # of Copies	2 bytes
(15) Print Disp	1 byte
(16) Prnt Mode Status	1 byte
(17 Print from Page	1 byte
(18) Print thru Page	1 byte
(19) Start Page #	1 byte
(20) First Header Pg	1 byte
(21) First Footer Pg	1 byte
(22) Bgn Footer Line	1 byte
(23) Page Length	1 byte
(24) Character Set 1	1 byte
(25) Character Set 2	1 byte
(26) Printer Number	1 byte
(27) Left Margin 1	1 byte
(28) Left Margin 2	1 byte
(29) Pitch	1 byte
(30) Format	1 byte
(31) Forms	1 byte
(32) Style	1 byte
(33) Summary	1 byte
(34) Lines	1 byte
(35) Reserved	4 bytes

Items 17 - 34 above are word processing document file print options. For other file types, values supplied for these items are ignored. However, the total space must be allocated.

(15) Print disposition
X'00' - Scratch
X'20' - Save
X'40' - Requeue

(16) Print mode status
X'00' - Spool
X'80' - Hold

(29) Pitch
X'01' - 10
X'02' - 12
X'03' - PS
X'04' - 15

(30) Format
X'00' - Unjustified
X'40' - With notes
X'80' - Justified

(31) Forms
X'00' - Continuous
X'20' - Form 2
X'40' - Form 1
X'80' - Standard

(32) Style
X'00' - Final
X'80' - Draft

(33) Summary
X'00' - Omit
X'80' - Print

(34) Lines
X'00' - 6 per inch
X'80' - 8 per inch

(b) Run Options

(12) Job Name	8 bytes
(13) Job Mode	1 byte
(14) Job Disposition	1 bytes
(15) Job Action	1 byte
(16) Job Class	1 byte
(17) CPU Time Limit	4 byte
(18) Dump Options	1 byte
(19) Reserved	11 bytes

(13) Job mode
X'00' - run
X'80' - hold

(14) Job disposition
X'80' - requeue

(15) Job action
X'00' - Warn
X'04' - Pause
X'80' - Cancel

(18) Dump options
X'00' - Program specified
X'C0' - Dump
X'80' - No dump

Stack On Output

```
                                                 Lower
             |                      |            Address
             |______________________|
    0(SP)    |                      |
             |  Return Code         |            Higher
             |______________________|            Address
             |      Preceding       |
             |     Stack Data       |
```

Output

A return code is placed in the top word of the stack.

Return Codes

Code	Description
0	Success.
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 and FDX2 do not agree.
36	VTOC error, FDX2 and FDR do not agree.
40	Invalid specification of file/library/volume.
44	VTOC unreliable.

Code	Description
48	System task not running, no spooled printing or noninteractive 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),
                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
+*                                          TOPWORD
+        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,
                JOBCLASS=(R5),CPULIMIT=(,'CANCEL'),DUMP=DUMPOPT
+LAB     PUSHA  0,MYPLIST                  POINT TO "PLIST" WITH STACK
+*                                         TOPWORD
+        MVI    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),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
```

4.2.72 SUBMIT - Submit Transmit or Retrieve Request (SVC 46)

Syntax

Format 1:

```
[label] SUBMIT {TRANSMIT}[,PLIST={(register)}]
               {RETRIEVE}        {   address }

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

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

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

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

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

                [,GROUP={(register)}][,STATUS={'ACTIVE'}]
                        { 'string' }          { 'HOLD' }
                        {  address }          {address }

                [,DISP={  'SAVE' })][,XFERDISP={'STORE'})]
                       {'SCRATCH'}             {'PRINT'}
                       { address }             { 'RUN' }

                [,PRNTMODE={'SPOOL'})][,PRTDISP={'SCRATCH'})]
                           {'HOLD' }'           {'REQUEUE'}
                           {address}            {  'SAVE' }
                                                { address }

                [,FORM#={(register)}][,COPIES={(register)}]
                        {  integer }          {  integer }
                        {  address }          {  address }

                [,JOBMODE={ 'RUN' }][,JOBDISP={'REQUEUE'}]
                          {  HOLD }           { address }
                          {address}
```

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

[,ACTION={ 'WARN' }][,JOBCLASS={(register)}]
         {'CANCEL'}            { 'string' }
         {'PAUSE' }            {  address }
         {address }

[,CPULIMIT={(register)}][,DUMP={ 'PROG'}]
           {address    }       {   'NO' }
                               { 'YES' }
                               {address}

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

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

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

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

[,START={(register)}][,FINISH={(register)}]
        {  integer }          {  integer }
        {  address }          {  address }

[,NUMBER={(register)}][,HEADER={(register)}]
         {  integer }          {  integer }
         {  address }          {  address }

[,FOOTER={(register)}][,LINE={(register)}]
         {  integer }        {  integer }
         {  address }        {  address }

[,LENGTH={(register)}][,CHARSET1={(register)}]
         {  integer }            {  integer }
         {  address }            {  address }

[,CHARSET2={(register)}][,PRINTER={(register)}]
           {  integer }           {  integer }
           {  address }           {  address }
```

```
[,MARGIN1={(register)}][,MARGIN2={(register)}]
          {  integer }          {  integer }
          {  address }          {  address }

[,PITCH={(register)}][,FORM={ (register) }]
        {   '10'   }        {'CONTINUOUS'}
        {   '12'   }        { 'STANDARD' }
        {   'PS'   }        {   'FORM1'  }
        {   '15'   }        {   'FORM2'  }
        {  address }        {   address  }

[,FORMAT={  (register) }][,SUMMARY={(register)}]
         {'UNJUSTIFIED'}           {  'OMIT'  }
         { 'JUSTIFIED' }           {  'PRINT' }
         {   address   }           {  address }

[,STYLE={(register)}][,LINES={(register)}]
        {  'FINAL' }         {    '6'    }
        {  'DRAFT' }         {    '8'    }
        {  address }         {  address }

[,PRTDISP={'SCRATCH'}][,COPIES={(register)}]
        {'REQUEUE'}            {  integer }
        {  'SAVE' }            {  address }
        { address }
```

Function

SUBMIT transfers files from one computer system to another over the WangNet communications link. If the initial parameter is TRANSMIT, SUBMIT requests the queuing of a VS file or a WP document for transferring to the target system. If the initial parameter is RETRIEVE, SUBMIT requests the queuing of a VS file or a WP document for retrieval from the specified location.

The following parameters are required for the specified operations. All others are optional.

- TRANSMIT VS file -- one of the following parameters is required:

  - FILENAME, LIBRARY, VOLUME and LOCATION
  - FILE and LOCATION
  - PLIST

- TRANSMIT VS WP document -- DOCID and LOCATION are required, unless PLIST is specified.

- RETRIEVE VS file -- one of the following parameters is required:

    - RFILENAME, RLIBRARY, RVOLUME and LOCATION
    - RFILE and LOCATION
    - PLIST

- RETRIEVE WP DOCUMENT -- DOCID and LOCATION are required unless PLIST is specified.

Restrictions

If the PLIST option is not used, the program issuing the SUBMIT must pop off the stack the additional 224 bytes that were pushed on the stack when the SUBMIT was issued.

Parameter Definitions

PLIST A 224-byte user-supplied parameter list (fullword aligned) for use by the SUBMIT SVC. If PLIST is specified, then the remaining parameters are optional and, if present, are used to modify the parameter list in place. The default values of any omitted parameters 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 parameters are used to build a parameter list on the stack. The default values of omitted parameters are used in this case. The user is responsible for popping off 224 bytes beyond the stack topword (SVC return code) on return. See the previous section, SUBMIT JOB OR PRINT REQUEST, for the structure of PLIST.

FILENAME The name of the file to be queued for transfer to or retrieval from a remote location.

LIBRARY The name of the library in which the file resides.

VOLUME The name of the volume on which the file resides.

FILE The name of the file (VS-OIS filename format) to be queued for transfer with the following format: '/VOLUME:LIBRARY.FILENAME/'.

RFILENAME For a TRANSMIT, the name of the file to be assigned on the remote system. RFILENAME is an optional parameter that is valid for only XFERDISP store option. For a RETRIEVE, this is the name of the file to be retrieved from the remote location.

RLIBRARY For a TRANSMIT, the name of the library to be assigned on the remote system in which the file resides. RLIBRARY is an optional parameter that is valid for only XFERDISP store option. For a RETRIEVE, this is the name of the library in which the file resides on the remote system.

RVOLUME For a TRANSMIT, the name of the volume to be assigned on the remote system on which the file resides. RVOLUME is an optional parameter that is valid for only XFERDISP store option. For a RETRIEVE, the name of the volume on which the file resides on the remote system.

RFILE For a TRANSMIT, the name of the file (VS-OIS file name format) to be assigned on the remote system with the following format: '/VOLUME:LIBRARY.FILENAME/'. RFILE is an optional parameter that is valid for only XFERDISP store option. For a RETRIEVE, the name of the file (VS-OIS file name format) to be retrieved from the remote location with the following format: '/VOLUME:LIBRARY.FILENAME/'.

DOCID For a TRANSMIT, the name of the WP document to be queued for transfer. The document name consists of a 4-digit document number, followed by the document library, which is designated by one lowercase or uppercase letter. For a RETRIEVE, the name of the WP document to retrieve from the remote location. This is an optional parameter that is valid for only XFERDISP store option.

PASSWORD For a TRANSMIT, the password for the WP document to transfer (if the document is password protected). For a RETRIEVE, the password given to the retrieved document (if the document is to be password protected, or if the document is to be protected with a new password if it was already password protected.) This is an optional parameter that is valid for only XFERDISP store option.

DOCVOL For a TRANSMIT, the name of the volume on which the document resides. For a RETRIEVE, the name of the volume on which the retrieved document will be placed. This is an optional parameter that is valid for only XFERDISP store option.

RDOCID For a TRANSMIT, the name of the WP document that the file is given at the remote location. The file name consists of a four-digit document number, followed by the document library, which is designated by one letter (uppercase or lowercase). This is an optional parameter that is valid for only XFERDISP store option. For a RETRIEVE, the name of the WP document to retrieve from the remote location.

RPASSWORD For a TRANSMIT, the password that protects the document at the remote location (if the document is to be protected by a new password). This is an optional parameter that is valid for only XFERDISP store option. For a RETRIEVE, the password of the document to be retrieved from the remote location.

RDOCVOL For a TRANSMIT, the name of the volume on which the document is placed at the remote location. This is an optional parameter that is valid for only XFERDISP store option. For a RETRIEVE, the name of the volume on which the document resides on the remote system.

LOCATION The name of the location to which or from which the file or document is to be transferred.

GROUP The name of the transfer group.

REPLACE The option to replace a duplicate file. If YES, the existing file is deleted providing the user has write access and the transfer completes normally. If NO, if a file with the same name exists, the transfer is aborted. NO is the default.

STATUS The initial status of the transfer. If ACTIVE, the request is scheduled at the earliest possible time. If HOLD, the request is held until released by either the user or the operator.

DISP If SAVE, the source file is not scratched after the transfer has completed. If SCRATCH, the source file is scratched upon completion of the transfer.

XFERDISP The transfer disposition determines whether or not the file or document is to be executed, printed or stored. If PRINT, the file or document is printed. If STORE, the file or document is stored. If RUN, the file is executed.

PRTCLASS The print class (A-Z) to be used.

FORM# The form number, in binary (0-255), to be used.

COPIES The number of copies, in binary, to be printed.

PRTDISP The disposition of the print file after printing. SCRATCH, REQUEUE or SAVE after printing.

PRNTMODE Initial status of this file when queued. SPOOL signifies that the file is eligible for printing upon submission of the request. HOLD signifies that the file is not eligible for print scheduling until released by the operator or the submitter.

JOBNAME The JOBNAME or blanks to be assigned to the job to be executed.

JOBMODE Initial status of job when queued. RUN signifies that the job is eligible for scheduling upon arrival at the specified location. HOLD specifies that the job is not eligible for scheduling until released by the operator or the submitter.

JOBDISP The disposition of the job after processing. REQUEUE after execution.

ACTION The system action to be taken if the CPU time limit expires. WARN specifies that a warning message is displayed on Workstation 0. CANCEL specifies that the job is to be force cancelled if the time limit expires. PAUSE signifies a forced pause or HELP if the CPU time limit expires.

JOBCLASS The job class (A-Z) of the job to be executed.

CPULIMIT CPU time limit in seconds that is stored as a binary value. If zero is supplied, no time limit is enforced.

START First page (in binary) of document to be printed.

FINISH Last page (in binary) of document to be printed.

NUMBER Starting page number (in binary) of the document.

HEADER Page number (in binary) for first header in the document.

FOOTER Page number (in binary) for first footer in the document.

LINE Line number (in binary) where footer begins.

LENGTH Page length (in binary) for the document.

CHARSET1 Printer character set (in binary) to be used in printing the document.

CHARSET2 Alternate character set (in binary) to be used in printing the document.

PRINTER Printer number (in binary) to be used to print the document.

MARGIN1 Left margin (in binary) to be used when printing the document.

MARGIN2 Alternate margin (in binary) to be used when printing the document.

PITCH Character pitch to be used when printing the document.

FORMAT Print format to be used when printing the document. Either unjustified, justified or with notes.

FORM Type of form to be used when printing the document.

STYLE Printing style to be used when printing the document.

SUMMARY Determines whether or not to print the document statistics.

LINES Determines printer lines per inch for the document.

Output

SUBMIT issues a return code in the stack top word that indicates the success, failure, or status of the operation.

Return Codes

Code	Description
0	Success.
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.
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 or volume.
44	VTOC unreliable.
48	System task not running.
52	Error during XMIT SVC.
56	Invalid options specified in parameter list (reserved bytes not zero).
60	Invalid document (document number or document library identifier.

64 Invalid remote document (document number or document library identifier.

68 Document volume not alphanumeric.

72 Invalid replace option.

76 Invalid status (not ACTIVE or HOLD).

80 Invalid disposition (not SAVE or SCRATCH).

84 Invalid transfer disposition.

88 Invalid file class (must be zero).

92 Invalid file type.

96 Invalid remote file type.

100 No access rights (REPLACE=NO and file exists or REPLACE=YES and no WRITE access)(RETRIEVE option).

104 Invalid location name.

108 No system task queue entry available.

112 GETMEM error during message port CREATE.

Example

```
 SEND     SUBMIT TRANSMIT,FILENAME='TRANSFILE',LIBRARY='DAILY',          -
                VOLUME='ACCOUNTS',LOCATION='SYSTEM'
+SEND     PUSHN 0,224              *    GET SPACE ON STACK FOR "PLIST"...
+         XC    0(224,15),0(15)    *    ... AND CLEAR IT TO ZEROES
+         MVC   0(8,15),*+10            SET FILE NAME
+         B     *+12                    BRANCH AROUND LITERAL
+         DC    CL8'TRANSFILE'          FILE NAME
+         MVC   8(8,15),*+10            SET LIBRARY NAME
+         B     *+12                    BRANCH AROUND LITERAL
+         DC    CL8'DAILY'              LIBRARY NAME
+         MVC   16(6,15),*+10           SET VOLUME NAME
+         B     *+10                    BRANCH AROUND LITERAL
+         DC    CL6'ACCOUNTS'           VOLUME NAME
+         MVI   142(15),1               SET LOCAL FILENAME  TYPE
+         MVC   144(8,15),*+10          SET LOCATION
+         B     *+12                    BRANCH AROUND LITERAL
+         DC    CL8'SYSTEM'             LOCATION
+*                                      (GROUP OPTION WILL GET DEFAULT
+*                                       VALUES)
```

```
+          MVI   192(15),X'00'         (REPLACE OPTION DEFAULTED TO
+*                                      "NO")
+          MVI   193(15),X'00'         (STATUS OPTION DEFAULTED TO
+*                                      "ACTIVE")
+          MVI   194(15),X'00'         (DISPOSITION OPTION DEFAULTED TO
+*                                      "SAVE")
+          MVI   195(15),X'00'         (TRANSFER DISPOSITION OPTION
+*                                      DEFAULTED TO "STORE")
+          PUSHA 0,0(,15)               POINT TO "PLIST" WITH STACK TOP
+*                                      WORD
+          MVI   0(15),3                FLAG REQUEST TYPE: 1 - JOB
+*                                                         2 - PRINT
+*                                                         3 - TRANSMIT
+*                                                         4 - RETRIEVE
+          SVC   46                      (SUBMIT)   ISSUE SVC
GET        SUBMIT RETRIEVE,RDOCID='0160A',LOCATION=(R5),DOCID='0063H'
+GET       PUSHN 0,224            *     GET SPACE ON STACK FOR "PLIST"...
+          XC    0(224,15),0(15)  *     ... AND CLEAR IT TO ZEROES
+          MVI   142(15),2              SET LOCAL FILENAME TYPE
+          MVI   143(15),2              SET REMOTE FILENAME TYPE
+          MVC   0(5,15),*+10           SET DOCUMENT ID
+          B     *+10                   BRANCH AROUND LITERAL
+          DC    CL5'0063H'             DOCUMENT ID
+          DS    0H
+*                                     (DOCUMENT NOT PASSWORD PROTECTED)
+*                                     (DOCUMENT VOLUME DEFAULTED)
+          MVC   71(5,15),*+10          SET REMOTE DOCUMENT ID
+          B     *+10                   BRANCH AROUND LITERAL
+          DC    CL5'0160A'             REMOTE DOCUMENT ID
+          DS    0H
+*                                     (REMOTE DOCUMENT NOT PASSWORD
+*                                      PROTECTED)
+*                                     (REMOTE DOCUMENT VOLUME DEFAULTED)
+          MVC   144(8,15),0(R5)        SET LOCATION
+*                                     (GROUP OPTION WILL GET DEFAULT
+*                                      VALUES)
+          MVI   192(15),X'00'         (REPLACE OPTION DEFAULTED TO
+*                                      "NO")
+          MVI   193(15),X'00'         (STATUS OPTION DEFAULTED TO
+*                                     "ACTIVE")
+          MVI   194(15),X'00'         (DISPOSITION OPTION DEFAULTED TO
+*                                     "SAVE")
+  MVI   195(15),X'00'                 (TRANSFER DISPOSITION OPTION
+*                DEFAULTED TO "STORE")
+  PUSHA 0,0(,15)                      POINT TO "PLIST" WITH STACK TOP
+*               WORD
+  MVI   0(15),4                       FLAG REQUEST TYPE:   1 - JOB
+*                                                          2 - PRINT
+*                                                          3 - TRANSMIT
+*                                                          4 - RETRIEVE
+  SVC   46   (SUBMIT)                 ISSUE SVC
```

```
      SUBMIT TRANSMIT,FILENAME=TRANSFLE,LIBRARY=LIBNAME,     -
             VOLUME='SYSTEM',LOCATION='MYSYS'
   +  PUSHN 0,224            *           GET SPACE ON STACK FOR "PLIST"...
   +  XC     0(224,15),0(15)   *         ... AND CLEAR IT TO ZEROES
   +  MVC    0(8,15),TRANSFLE            SET FILE NAME
   +  MVC    8(8,15),LIBNAME             SET LIBRARY NAME
   +  MVC    16(6,15),*+10               SET VOLUME NAME
   +  B      *+10    BRANCH AROUND LITERAL
   +  DC     CL6'SYSTEM'                 VOLUME NAME
   +  MVI    142(15),1                    SET LOCAL FILENAME TYPE
   +  MVC    144(8,15),*+10              SET LOCATION
   +  B      *+12    BRANCH AROUND LITERAL
   +  DC     CL8'MYSYS'                  LOCATION
   +*               (GROUP OPTION WILL GET DEFAULT
   +*               VALUES)
   +  MVI    192(15),X'00'               (REPLACE OPTION DEFAULTED TO
   +*                                     "NO")
   +  MVI    193(15),X'00'               (STATUS OPTION DEFAULTED TO
   +*                                     "ACTIVE")
   +  MVI    194(15),X'00'               (DISPOSITION OPTION DEFAULTED TO
   +*                                     "SAVE")
   +  MVI    195(15),X'00'               (TRANSFER DISPOSITION OPTION
   +*                                     DEFAULTED TO "STORE")
   +  PUSHA 0,0(,15)                     POINT TO "PLIST" WITH STACK TOP
   +*               WORD
   +  MVI    0(15),3                    FLAG REQUEST TYPE: 1 - JOB
   +*                                                      2 - PRINT
   +*                                                      3 - TRANSMIT
   +*                                                      4 - RETRIEVE
   +         SVC   46   (SUBMIT)         ISSUE SVC
```

4.2.73 SYSERROR - System Error Code Definitions

Syntax

    SYSERROR

Function

Establishes symbolic names and their equivalent numeric codes for common system error conditions.

Example

```
          SYSERROR
+SYSERROR        DSECT
+* SYSTEM ERROR CODE DEFINITIONS
+*
+SYSEREND         DS  CL1
+@ERSUCC     EQU 0                  SUCCESS
+@ERIPVAL    EQU 1                  ILLEGAL PARAMETER VALUE
+@ERIPTYP    EQU 2                  ILLEGAL PARAMETER TYPE
+@ERPROT     EQU 3                  ATTEMPTED PROTECTION VIOLATION
+@ERUNPRIV   EQU 4                  UNPRIVILEGED CALLER
+@ERTHEAP    EQU 5                  TASK HEAP EXHAUSTED
+@ERSHEAP    EQU 6                  SYSTEM HEAP EXHAUSTED
+@ERVNM      EQU 7                  VOLUME NOT MOUNTED
+@ERVACC     EQU 8                  VOLUME ACCESS DENIED
+@ERFDE      EQU 9                  FILE DOES NOT EXIST
+@ERACC      EQU 10                 FILE ACCESS DENIED
+@EROPQ      EQU 11                 OPEN FILE QUOTA EXHAUSTED
+@ERLLQ      EQU 12                 LINK LEVEL QUOTA EXHAUSTED
+@ERSUBQ     EQU 13                 SUBTASK QUOTA EXHAUSTED
+@ERVEO      EQU 14                 VOLUME EXCLUSIVELY OPENED
+@ERBIU      EQU 15                 ALL BUFFERS IN USE
+@ERDDE      EQU 16                 DIRECTORY DOES NOT EXIST
+@ERFIU      EQU 17                 FILE IN USE
+@ERIOERR    EQU 18                 I/O ERROR
+@ERDINFO    EQU 19                 DISK INFORMATION MISMATCH
+@ERIPROG    EQU 20                 INVALID PROGRAM FILE
+@ERNOTIME   EQU 21                 NO INTERVAL DEFINED
+@ERVIU      EQU 22                 VOLUME IN USE
+@ERUNEXP    EQU 23                 UNEXPIRED FILE
+@ERFILENAME EQU 24                 ILLEGAL FILENAME
+@ERHNAME    EQU 26                 HEAP NAME ERROR
+@ERTID      EQU 27                 ILLEGAL TASK ID
+@ERDEV      EQU 28                 ILLEGAL DEVICE
+@ERFILTYP   EQU 29                 ILLEGAL FILE TYPE
+@ERINV      EQU 30                 ILLEGAL OPTIONS COMBINATION
+@ERDOCNAME  EQU 31                 ILLEGAL DOCUMENT NAME
+@ERSYS      EQU 32                 SYSTEM TASK ERROR
+@ERNODIAG   EQU 33                 NO DIAGNOSTIC PAGES AVAILABLE
```

```
+@ERDIAGUSE  EQU 34                 DIAGNOSTIC PAGES USE ERROR
+@ERVOLNAME  EQU 35                 ILLEGAL VOLUME NAME
+@ERVAM      EQU 36                 VOLUME ALREADY MOUNTED
+@ERDTYP     EQU 37                 ILLEGAL DEVICE TYPE
+@ERRES      EQU 38                 DEVICE RESERVED

+@ERMTYPE    EQU 39                 MEDIA TYPE ERROR
+@ERVLM      EQU 40                 VOLUME LABEL MISMATCH
+@ERVTM      EQU 41                 VOLUME TYPE MISMATCH
+@ERADDRS    EQU 42                 ADDRESSING TYPE ERROR
+@ERDETACH   EQU 43                 DEVICE IS DETACHED
+@ERNOTPROG  EQU 44                 DEVICE IS NOT PROGRAMMABLE
+@ERFNOTEXC  EQU 45                 FILE NOT EXCLUSIVELY OPENED
+@EROPEN     EQU 46                 ERROR ON OPEN
+@ERVND      EQU 47                 VOLUME NOT DISMOUNTABLE
+@ERTIMEOUT  EQU 48                 TIMEOUT ON REQUEST
+@ERNPF      EQU 49                 ERROR FROM @PROC@
+@ERVRES     EQU 50                 VOLUME RESERVED
+@ERDEVNUM   EQU 51                 NUMBER OF DEVICES NOT AVAILABLE
+@ERDEVNAME  EQU 52                 ILLEGAL DEVICE NAME
+@ERFOPEN    EQU 53                 FILE NOT OPEN
+@ERTASK     EQU 54                 TASK IN PROGRESS
+@ERTASKCR   EQU 55                 TASK CREATE/DELETE ERROR
+@ERDEVACT   EQU 56                 DEVICE ACTIVE
+@ERNOTRES   EQU 57                 DEVICE NOT RESERVED
+@ERSPACE    EQU 58                 NO SPACE ON VOLUME
+@ERNOCODE   EQU 59                 PP NOT LOADED
+@ERBUSY     EQU 60                 IOP BUSY
+@ERVSPACE   EQU 61                 NOT ENOUGH VIRTUAL SPACE
+@ERFILMAP   EQU 62                 FILE ALREADY MAPPED/NOT MAPPED
+@ERMODECON  EQU 63                 MODE CONFLICT WITH PREVIOUS OPEN
+@ERMODENS   EQU 64                 MODE NOT SUPPORTED ON THIS DEVICE
+@EREOF      EQU 65                 I/O ATTEMPTED PAST END OF FILE
+@ERNOTSUP   EQU 66                 SVC NOT SUPPORTED ON THIS SYSTEM
+@ERCLOSE    EQU 67                 ERROR ON CLOSE
+@ERDIRACC   EQU 68                 DIRECTORY ACCESS ERROR (CBAM)
+@ERNDE      EQU 69                 NODE DOES NOT EXIST
+@ERDUP      EQU 70                 DUPLICATE
+@ERDIRFULL  EQU 71                 NO MORE INDICES IN DIRECTORY
+@ERINS      EQU 72                 INSERT FAILED
+@ERPATH     EQU 73                 INVALID PATH NAME
+@ERINP      EQU 74                 ILLEGAL NUMBER OF PARAMETERS
+@ERMISALIGN EQU 75                 PARAMETER NOT PROPERLY ALIGNED
+@ERVNAME    EQU 76                 VOL MOUNTED, BUT NAME DIFFERS
+@ERUREJECT  EQU 77                 USER REJECTED OPERATION
+@ERINSUFMBO EQU 78                 INSUFFICIENT MAILBOX RESERVE
+@ERINSUFPCN EQU 79                 PROCESS CHILD QUOTA EXHAUSTED
+@ERINSUFQ   EQU 80                 QUOTA EXHAUSTED
+@ERINSUFMEM EQU 81                 INSUFFICIENT MEMORY
+@ERNOTFOUND EQU 82                 ARGUMENT OR VALUE NOT FOUND
```

```
+@ERTRUNC     EQU 83                 Value truncation detected
+@ERNAE       EQU 84                 NODE ALREADY EXISTS
+@ERTERM      EQU 85                 TERMINAL NODE
+@ERADE       EQU 86                 ATTRIBUTE DOES NOT EXIST
+@ERPARAMNOT  EQU 87                 REQUIRED PARAMETER NOT FOUND
+@ERSERVNOTA  EQU 88                 SYSTEM SERVICE NOT AVAILABLE YET
+@ERSTRTRUNC  EQU 89                 String truncation detected
+@ERNVSEDIR   EQU 90                 VSE DIRECTORY NOT PRESENT ON VOL
+@ERDIRIOER1  EQU 91                 DIRECTORY I/O ERROR  COPY 1
+@ERDIRIOER2  EQU 92                 DIRECTORY I/O ERROR  COPY 2

+@ERDIRUNREL  EQU 93                 DIRECTORY UNRELIABLE:MULT I/OERR
+@ERDIRSPEXH  EQU 94                 DIRECTORY SPACE ON VOL EXHAUSTED
+@ERBADIOCOM  EQU 95                 BAD IO COMMAND ISSUED
+@ERRUNPRIV   EQU 96                 ATTEMPT TO RUN PRIV CODE WHILE UNPRIV
+@ERTRANFUL   EQU 97                 BAD BLOCK TRANSLATION TABLE FULL
+@ERBOUNDARY  EQU 98                 Process level boundary/EOstack found
+@ERMODULE    EQU 99                 Module frame encountered
+@ERNOTINMOD  EQU 100                Signal invalid outside module
+@ERINVGARG   EQU 101                Invalid Getarg call
+@ERPBLANK    EQU 102                DATA = BLANKS OR GARBAGE
+@ERBADPID    EQU 103                PROCESS ID NOT FOUND
+@ERNOTPAUSED EQU 104                PROCESS NOT PAUSED
+@ERBGPROC     EQU 105               PROCESS LIVES IN BACKGROUND
+@ERMAXTRAPS   EQU 106               TOO MANY TRAPS SPECIFIED
+@ERBUFEMPTY   EQU 107               Buffers empty
+@EROUTRANGE   EQU 108               OUT OF RANGE
+@ERDBGNOTACT  EQU 109               No debugging active for this task
+*
+* Security Logging Return Codes:
+*
+@ERGETSETEVENTS    EQU    120        Can't do both Get and Set Events
+*                                     on same CNTROLOG call
+@ERGETRSTEVENTS    EQU    121        Can't do both Get and Reset Events
+*                                     on same CNTROLOG call
+@ERGETSETVIOLS     EQU    122        Can't do both Get and Set Violations
+*                                     on same CNTROLOG call
+@ERGETRSTVIOLS     EQU    123        Can't do both Get and Rst Violations
+*                                     on same CNTROLOG call
+@ERSTATCNTRL       EQU    124        Can't do both Control and Getstatus
+*                                     on same CNTROLOG call
+@ERINACTNOTNEW     EQU    125        Can't request Inactfile when not
+*                                     doing a Newlog on CNTROLOG call
+@ERNEWLIBNOTNEW    EQU    126        Can't specify Newlib and Control
+*                                     if Control not = Newlog
+@ERLOGGINGON       EQU    127        Logging is already active
+@ERLOGINACTIVE     EQU    128        Logging is not active
+@ERNOREPLY         EQU    129        No reply message from Systsk
+@ERWRONGMSG        EQU    130        Invalid message sent back by Systsk
+@ERLOGNOTPRIV      EQU    131        Caller not authorized to log this
+*                                     Event Type
```

```
+@ERLOGEVNTNOTSET  EQU   132       Event specified on PUTLOG is not
+*                                  set to be logged
+@ERLOGVIOLNOTSET  EQU   133       Violation specified on PUTLOG is
+*                                  not set to be logged
+@ERNOLOGGING      EQU   134       Logging task has been terminated
+*

+*  Synchronous object error codes
+*
+*
+@ERALRDYHAS     EQU  135          User already has this sync object
+@ERSOUNAV       EQU  136          Sync object unavailable (NOWAIT opt)
+@ERSOACCDIS     EQU  137          Sync object access disallowed
+@ERSONOTOWN     EQU  138          Sync object not owned by caller
+@ERMKDEL        EQU  139          Sync object marked for delete
+*
```

```
+* Errors for Font Services
+@ERFTFDE       EQU   283          Font entry doesn't exist
+@ERFTNONE      EQU   284          No fonts installed for the device
+@ERFTNUM       EQU   285          Font number already exists
+*
+*         File Format Manager Errors
+@ERUSERID       EQU  286        * User ID invalid on remote system
+@ERUSRLST       EQU  287        * Unable to read USERLIST
+@ERUNKWN        EQU  288        * Unable to INVOKE a File Server
+*
+SYSERLENGTH      EQU *-SYSERROR
+         CSECT
```

4.2.74 TCOPTION - Set Telecommunications Stream Options

Syntax

```
[label] TCOPTION    UFB={(register)}[,STREAM={READER} ]
                        { address  }         {PUNCH }
                                             {PRINTER}

                [,DEVTYPE={2780  }][,RECSIZE=integer]
                          {3780  }
                          {TCDIAG}

                [,COMP={YES}][,PRINT={NO }][,BLOCKED={YES}]
                       {NO }         {YES}           {NO }

                [,TRANSMISSION=({TRANSPARENT    ,}
                                {NONTRANSPARENT,}

                                { BLOCKED, }
                                {UNBLOCKED,}

                                {UNPADDED, }
                                {PADDED,   }

                                {COMPRESSED,  }
                                {UNCOMPRESSED,}

                                {EBCDIC})]
                                {ASCII }
```

Function

Sets the telecommunications (TC) stream options in the user file block (UFB). The UFB TC stream options consist of the data option, the transmit/receive option, and the maximum record size option. 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
  - Bit 1 = 1 Compressed VS record format
  - Bit 2 = 1 Blocked VS record format
  - Bits 3-5 Reserved
  - Bits 6-7 = 00 For card reader stream
    = 01 For card punch stream
    = 10 For printer stream
    = 11 Reserved

- TC transmit/receive option:
  - Bit 0 = 1 Perform code translation from EBCDIC to ASCII
  - Bit 1 = 1 Compress transmitted record data
  - Bit 2 = 1 Pad transmitted records to exact length with space codes
  - Bit 3 = 1 Block transmitted records
  - Bit 4 = 1 Transmit in transparent mode
  - Bits 5-7 Reserved

The third byte in the TC stream option is equal to the maximum or exact transmitted record length minus one.

Parameter Definitions

UFB — The address of a user file block (UFB), which may be supplied as a register specified in parentheses that contains the UFB address, or as an address 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 card reader, PUNCH for card puncher, 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 is set to 1; otherwise, the bit is set to 0.

BLOCKED — If NO, the corresponding bit in the data option is set to 0; otherwise, to 1.

COMP — If NO, the corresponding bit in the data option is set to 0; otherwise to 1.

TRANSMISSION — The bits in the transmit/receive option are set according to the parameter specified. For example, if TRANSPARENT is specified, the corresponding bit 4 in the transmit/ receive option is set to 1; if NONTRANSPARENT is specified, bit 4 in the transmit/receive option is set to 0.

RECSIZE — The third byte in the TC stream option is set to the integer value minus 1.

Example

```
LAB1     TCOPTION UFB=(R2),STREAM=PUNCH,BLOCKED=NO,RECSIZE=10,
              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
```

4.2.75 TIME - Get Date and Time (SVC 2)

Syntax

```
[label] TIME     [JUL][,HMS]
                 [YMD][,CLK]
```

Function

The current date and time are returned to the calling routine in one of two forms. The date is returned in either Julian or year-month-day format and the time is returned in hours-minutes-seconds or clock ticks format.

Parameter Definitions

JUL — The date is returned in Julian format in the higher-addressed word of the two-word area pushed onto the stack as follows: A packed number in the format 00YYDDDF, where YY is the year, DDD is the day of the year, and F a hexadecimal 'F' (positive sign).

YMD — The date is returned in the standard format in the higher-addressed word of the two-word area pushed onto the stack as follows: A packed number in the format 0YYMMDDF, where YY is the year, MM is the month, DD is the day, and F a hexadecimal 'F'.

HMS — 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 in the format 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 of second in tenth of second

The minimum time value is 00000000; the maximum is 23595999.

CLK — 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.

Stack On Input

```
                                          Lower
             |                      |     Address
             |______________________|
  0(SP)      |                      |
             | (1) Requested Time   |
             |_______Format_________|
  4(SP)      |                      |
             | (2) Requested Date   |     Higher
             |_______Format_________|     Address
             |                      |
             | Preceding Stack Data |
```

(1) Requested time format:
    0 = Decimal format, hours, minutes, seconds, CSECs
    1 = Binary format, CSECs

(2) Requested date format:
    0 = JUL (Julian); year, day-in-year
    1 = YMD; year, month, day

Stack On Output

```
                                          Lower
             |                      |     Address
             |______________________|
  0(SP)      |                      |
             |  (1)   Time          |
             |______________________|
  4(SP)      |                      |
             |  (2)   Date          |     Higher
             |______________________|     Address
             |                      |
             | Preceding Stack Data |
```

(1) Time
    Decimal - Time of day in hours, minutes, seconds, CSECs
    Binary - Time of day in CSECs

(2) Date in one of two formats:
    YMD - packed, in the format 0YYMMDDF, where
        YY is the year
        MM is the month
        DD is the day
        F is a hexadecimal F for unpacking
    JUL - Packed, in the format 00YYDDDF, where
        YY is the year
        DDD is the day of the year (1-365)
        F is a hexadecimal F for unpacking

Examples

```
          TIME JUL
+         PUSHA 0,0
+         PUSHA 0,0
+         SVC   2 (TIME)

          TIME YMD,CLK
+         PUSHA 0,1
+         PUSHA 0,1
+         SVC   2 (TIME)
```

4.2.76 TPLAB - Describe Magnetic Tape File Header, Trailer and End-of-Volume Labels

Syntax

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

Function

Describes the magnetic tape file header, trailer, and end-of-volume labels in ANSI standard format.

Parameter Definitions

NODSECT — Specification of NODSECT results in the TPLAB fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, a DSECT with the name TPLAB (plus optional SUFFIX) is generated.

REG — Provides for the optional specification of a register for which a USING statement for the TPLAB fields is generated.

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

Structure

TPLAB
BEGIN

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+0	ID			
+4	FILE			
+8				
+C				
+10				
+14		VOL1SER		
+18				FILESECTION
+18				SECTION
+1C				FILESEQ
+20				GENERATION
+24				VERSION
+28		CREATION		
+2C				EXPIRATION
+30				
+34		ACCESS	BLKCOUNT	
+38				
+3C	SYSTEM			
+40				
+44				
+48		CREATOR		
+4C	SPARE1			

LENGTH = 50

Example

```
          TPLAB REG=5
+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 *
+TPLABID              DS CL4        'HDR1', 'EOV1', OR 'EOF1'
+TPLABFILE            DS CL17       UP TO 17 ASCII CHARACTERS,
+*                                  LEFT ADJUSTED AND PADDED
+*                                  WITH BLANKS, NAMING THE FILE
+TPLABVOL1SER         DS CL6        VOLUME SERIAL NUMBER MATCHIN
+*                                  'VOL1SER' IN VOLUME LABEL (OF THE
+*                                  FIRST VOLUME, IF A MULTIVOLUME
+*                                  FILE)
+TPLABFILESECTION     DS CL4'0001'  ORDER OF VOLUME IN A MULTI-
+*                                  VOLUME FILE (ASCII '0001' FOR A
+*                                  SINGLE-VOLUME FILE)
+TPLABFILESEQ         DS CL4        FILE SEQUENCE NUMBER
+*                                  ON MULTIFILE VOLUME (1ST FILE
+*                                  IS ASCII '0001')
+TPLABGENERATION      DS CL4'0001'  GENERATION NUMBER (CURRENTLY
+*                                  ALWAYS '0001', USE DEFERRED)
+TPLABVERSION         DS CL2'00'    VERSION IN GENERATION,
+*                                  CURRENTLY ALWAYS ZERO
+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)
+TPLABEXPIRATION      DS CL6        EXPIRATION DATE IN THE
+*                                  ABOVE FORMAT
+TPLABACCESS          DS CL1' '     ACCESS PROTECTION (FILE
+*                                  PROTECTION CLASS OR BLANK)
+TPLABBLKCOUNT        DS CL6        BLOCK COUNT IN TRAILER LABEL
+*                                  AS SIX ASCII DIGITS. ALWAYS
+*                                  PLACED IN 'EOV1' AND 'EOF1'
+*                                  LABELS. ASCII ZEROS IN HDR LABEL.
+TPLABSYSTEM          DS CL13       CHARACTERS IDENTIFYING
+*                                  THE CREATING SYSTEM
+TPLABCREATOR         DS CL3        FILE CREATOR ID OR BLANKS
+TPLABSPARE1          DS CL4        RESERVED - MUST BE BLANKS
+TPLABEND             EQU *
+TPLABLENGTH          EQU TPLABEND-TPLABBEGIN
+BEGIN    CODE
+         USING TPLAB,5
```

4.2.77 TPLB2 - Describe Magnetic Tape Secondary Header, Trailer, and End-of-Volume Labels

Syntax

```
    TPLB2    [NODSECT][,REG=expression][,SUFFIX=character]
```

Function

Describes the structure of a secondary magnetic tape header, trailer and end of volume label in ANSI standard format.

Parameter Definitions

NODSECT — Specification of NODSECT results in the TPLB2 fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, a DSECT with the name TPLB2 (plus optional SUFFIX) is generated.

REG — Provides for the optional specification of a register for which a USING statement for the TPLB2 fields is generated.

SUFFIX — If provided, all labels are generated by the concatenation of the letters TPLB2, the user-provided SUFFIX (one ASCII character in length), and the field name.

Structure

TPLB2

BEGIN

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+0	ID			
+4	RECFM	BLKI		
+8			RECI	
+C				ORG
+10	OIS1	OIS2	OIS3	OISSYS
+14	OISBLS	SPARE1		
+18				
+1C				
+20				
+24				
+28				
+2C				
+30			BOFF	
+34	SPARE2			
+38				
+3C				
+40				
+44				
+48				
+4C				

LENGTH = 50

Example

```
         TPLB2 REG=2,SUFFIX=L
+TPLB2L         DSECT
+*
+*        MAGNETIC TAPE SECONDARY HEADER, TRAILER, AND END OF
+*        VOLUME LABELS CONFORM TO ANSI STANDARDS, AS FOLLOWS
+*
+*        DATE 8/28/81
+*        VERSION 5.01.01
+*
+TPLB2LBEGIN         EQU *
+TPLB2LID            DS CL4        'HDR2', 'EOV2', OR 'EOF2'
+TPLB2LRECFM         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
+TPLB2LBLKL          DS CL5        BLOCK LENGTH (ASCII)
+TPLB2LRECL          DS CL5        RECORD LENGTH (ASCII)
+TPLB2LORG           DS BL1        FILE ORGANIZATION
+TPLB2LORGCONSEC     EQU X'01'     CONSECUTIVE
+TPLB2LORGWP         EQU X'04'     WP FILE
+TPLB2LORGPRINT      EQU X'40'     PRINT FILE
+TPLB2LORGPROG       EQU X'80'     PROGRAM FILE
+TPLB2LOIS1          DS BL1        RESERVED FOR VS/OIS FILE TRANSFER
+TPLB2LOIS2          DS BL1        RESERVED FOR VS/OIS FILE TRANSFER
+TPLB2LOIS3          DS BL1        RESERVED FOR VS/OIS FILE TRANSFER
+TPLB2LOIS3PLOG      EQU X'01'     FILE PROLOGUE SECTOR PRESENT
+TPLB2LOISSYS        DS BL1              OIS SYSTEM INDICATOR
+TPLB2LOISSYS100     EQU X'01'           OIS-100 FILE
+TPLB2LOISSYS200     EQU X'02'           OIS-200 FILE
+TPLB2LOISBLS        DS BL1              OIS BYTES IN LAST SECTOR
+TPLB2LSPARE1        DS CL29             RESERVED FOR OPERATING
+*                                           SYSTEM USE
+TPLB2LBOFF          DS CL2        BUFFER OFFSET
+TPLB2LSPARE2        DS CL28       RESERVED - MUST BE ASCII
+TPLB2LEND           EQU *
+TPLB2LLENGTH        EQU TPLB2LEND-TPLB2LBEGIN
+BEGIN    CODE
+         USING TPLB2L,2
```

4.2.78 TRANSMIT - Transmit Telecommunications I/O (SVC 3)

Syntax

Format 1:

```
    [label] TRANSMIT    DATA,OFB=    {  address },
                                     {(register)}

                    COMMAND=         {      address     },
                                     {    (register)    }
                                     {self-defining term}

                    RECAREA=         {  address },
                                 {(register)}

                    LENGTH=          {      address     },
                                     {    (register)    }
                                     {self-defining term}

                    IOCWOPTS=        {  address }
                                     {(register)}
```

Format 2:

```
    [label] TRANSMIT    CONTROL,OFB={  address },
                                     {(register)}

                        COMMAND=     {      address     },
                                     {    (register)    }
                                     {self-defining term}

                        IOCWOPTS=    {  address }
                                     {(register)}
```

Function

TRANSMIT DATA initiates a WRITE SIO operation directed to the DLP on the addressed communication channel device. TRANSMIT CONTROL initiates a CONTROL SIO operation directed to the DLP on the addressed communication channel device.

The TRANSMIT macroinstruction invokes the XIO SVC (SVC 3) to initiate the I/O operation. The XIO SVC checks that the specified communication channel is opened, and that the communication channel and the DLP are not reserved by another task. A CHECK for completion of the I/O operation is not implicit, and must be produced by waiting for reception of the IOSW using the TCIO option of the CHECK facility. See the XIO macro for further information.

Parameter Definitions

OFB

A required parameter that defines the address of the open file block (OFB) for the VS-DLP I/O channel device to be used in the I/O operation. The address can be obtained from the interprocessor control block (IPCB). Specified as an address expression that points to a 4-byte field which contains the OFB address in its low-order three bytes, or as a register in parentheses that contains the address of the OFB in the low-order three bytes.

COMMAND

A required parameter which enables the user to supply a value for the command byte (byte 0) of the IOCW constructed by the XIO SVC. The default for the TRANSMIT DATA option is X'80'. The default for the TRANSMIT CONTROL option is X'C0'. The parameter value may be specified as an address expression that points to a 1-byte field which contains the command byte, as a register in parentheses that contains the command byte in its least significant byte, or a character string which is the command byte.

RECAREA

A required parameter that defines the address of the area which contains the data to be transferred. The value specified in the RECAREA parameter is the value placed by the XIO SVC in the data address field of the IOCW (bytes 1-3). The parameter can be specified as an address expression, or as a register in parentheses which contains the address of the reception area in the low-order three bytes.

LENGTH

A required parameter that defines the length of the data to be transferred in the I/O operation. The value specified is the value placed by the XIO SVC in the data count field of the IOCW (bytes 4-5). This parameter can be specified an address expression that points to a 2-byte area which contains in binary the length in bytes, a register in parentheses that contains the length in bytes in its low-order two bytes, or a character string in single quotes which is the length in bytes.

IOCWOPTS

Optional nonzero values for the last three bytes (bytes 6-8) of the 9-byte IOCW may be supplied with the IOCWOPTS parameter. The last three bytes default to zeroes. This parameter can be specified as an address expression that points to a 3-byte field which contains the option bytes, or as a register in parentheses that contains the three option bytes in its low-order three bytes.

Output

A return code is issued by the XIO SVC in the stack top word, as follows:

- Low-order halfword of return code field - binary return codes.
- High-order halfword of the return code field contains the residual block counts:
  - For return codes 4, 8 - specified block size minus number of bytes actually read or written.
  - Other return codes - always zero.

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 other return codes are set, the operation has been suppressed.

Return Codes

Code	Description
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 be either in the 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 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 opened file in WPSHARE mode, file not locked.

44 READ XIO attempted to opened file in WPSHARE mode, file locked by another user.

Example

```
 GO      TRANSMIT  CONTROL,OFB=CTRLBLK
+GO      PUSHA  0,0                  CLEAR IOCW OPTIONS AREA
+        PUSHA  0,0                  CLEAR NEXT 4 BYTES OF SPACE
+        MVI    1(15),8              SET DATA LENGTH TO 8
+        PUSHA  0,0                  CLEAR DATA TRANSFER ADDRESS
+        MVI    0(15),192            SET DEFAULT IOCW COMMAND CODE
+        PUSHA  0,0                  SPACE FOR OFB ADDRESS  01\
+        MVC    0(4,15),CTRLBLK      PUSH ADDRESS OF THE "OFB" 01\
+        MVI    0(15),X'01'          MARK AS 'TC XIO'
+        SVC    3  (XIO)
```

4.2.79 UFB - Describe User File Block (UFB)

Syntax

UFB [NODSECT][,REG=expression][,SUFFIX=character]

Function

Describes the structure of a user file block (UFB). A user file block must be present in the user's modifiable program area before opening a file. The address of this block is stored in the open file block (OFB) by the OPEN SVC and the address of the OFB is placed in this block.

One UFB control block must exist for every file that a user program accesses. The OPEN SVC uses this structure to generate necessary system control blocks which control access to the file. Once a file is opened, the user program may read and write to the file through the use of DMS routines. Once access is no longer needed, the file must be closed by issuing the CLOSE SVC.

Parameter Definitions

NODSECT Specification of NODSECT results in the UFB fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, a DSECT with the name UFB (plus optional SUFFIX) is generated.

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

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

Structure

UFB

		BYTE 0	BYTE 1	BYTE 2	BYTE 3
BEGIN	0	VECT			
	+4				
	+8				
	+C				
	+10				

ORG

```
V     |      0  | VREAD                                      |
            +4  | VWRITE                                     |
            +8  | VREWRITE                                   |
            +C  | VDELETE                                    |
           +10  | VSTART                                     |
                  BYTE 0     BYTE 1     BYTE 2     BYTE 3
```

ORG

```
            +4  | FLAGSD    |                                |
            +8  |                                            |
            +C  |                                            |
           +10  |                                            |
           +14  | ERRAD                                      |
           +18  | EODAD                                      |
           +1C  | RECAREA                                    |
           +20  | KEYAREA                                    |
           +24  | FS1       |FS2        | BLKSIZE            |
           +28  | RECSIZE               | FORG      | F1     |
           +2C  | F2        | DEVCLASS  | FLAGS     | DEVADDR|
F3    |    +30  | PRTCLASS  |FORMNO     | PRNAME             |
```

ORG

```
           +30  | SLOTSIZE              |                    |
           +34  |                                            |
           +38  |                       | VOLSER             |
           +3C  |                                            |
           +40  | DIRNAME                                    |
           +44  |                                            |
           +48  | FILENAME                                   |
           +4C  |                                            |
           +50  | FPCLASS   | CREATOR                        |
ALTCNT |   +54  | ALTPTR                                     |
```

ORG

```
           +54  | LOGRECCNT                                  |
```

ORG

```
           +54  | RELPOS                                     |
```

ORG

```
           +54  | MCTYPE    |TCDATAOPT|TCXMITOPT|TCMAXRECSZ|
```

ORG

```
          +54  |          | WPAID                              |

                BYTE 0     BYTE 1     BYTE 2     BYTE 3
```

ORG

```
          +58  | F4       | NRECS                              |
          +5C  | LRECSAVE            | RETPD                   |
          +60  | BCB1                                          |
          +64  |                                               |
          +68  |                                               |
          +6C  |                                               |
```

ORG

```
XIOFLAGS| +60  | OFB                                           |
BUFCMD  | +64  | BUFADR                                        |
          +68  | BUFDATAL            | BUFOFFSET               |
          +6C  | BUFBLOCK                       | BCBFLAGS     |
```

ORG

```
          +68  | TIMEEXIT                                      |
          +6C  | HOLDID                         | TIME         |
```

ORG

```
          +68  | SHROPNCODE                                        |
          +6C  | SHROPNRCSZ          | SHROPNFORG | SHROPNSPARE    |
          +70  | BUFSIZE             |  CHKSIZE                    |
          +74  | RES3                                              |
```

ORG

```
          +
          +74  | XDATE                                         |
```

ORG

```
          +74  | OUTRECS                                       |
```

ORG

```
          +78  | SPB                            | NBLKS        |
```

ORG

```
          +74  |                                | DMSGID       |
          +78  |                     |  MAXTFR                 |
```

ORG

+78			RES1	OPFLAGS
+7C	LF	LFMOD		

ORG

+7C		XCODE	EREC	
+80	VERSION	EBLK		
+84	BUFSTART			
+88	RDLTH		PRTCOPIES	

ORG

+88			WPBLKSIZE	WPBLS

ORG

+84	PTRB			
+88	PTRC			

ORG

+88	SFBVERSION			

Indexed Disk File Extension Section

	BYTE 0	BYTE 1	BYTE 2	BYTE 3
+8C	KEYPOS		KEYSIZE	GKSIZE
+90	HXBLK			DABLK
+94			PKI	
+98	PTRD			
+9C	PTRI			
+A0	PTRN			

ORG

+90	SHRAXD1			
+94				
+98				
+9C				
+A0				
+A4	BCBIOUT			
+A8				
+AC				
+B0				

ORG

+A4	BIRECAREA			
+A8	BIRECSIZE		BIAXD1MASK	
+AC				
+B0				
+B4	PKD		SPAREINX	

DMS/TX Disk File Extension Section

		BYTE 0	BYTE 1	BYTE 2	BYTE 3
	+B8	DXOM	DXRECBLK	DXDBNAME	
	+BC				
DXFV#	+C0	DXFV#SEQ#			
	+C4	DXFV#DT			
	+C8				
	+CC	DXRECO	DXLSBREC	DXREORGLF	DXCS
	+D0	DX#BIJS		DXFLAGS	DXPSBBUF
	+D4			DXSPARE	
	+D8	DXMSB			

Magnetic Tape File Extension Section

ORG

+8C	TSPARE1			
+90	TBCB			
+94				
+98				
+9C				
+A0	TLABELS	TDEN	TSEQ	
+A4	TFLG	TVOLSEQ	TSAVEVOL	
+A8				
+AC	TPARITY	TSPARE2		
+B0				
+B4				

Example

```
          UFB REG=(R2),SUFFIX=A
+UFBA         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  03-18-83
+*        VERSION 6.00.32
+*
+********************************************************************
+UFBABEGIN                  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.
+* ***********************************************************
+UFBAVECT                   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.
+                   ORG UFBAVECT
+UFBAV                      DS  0F          (PREFIX TO EQUATE LABELS)
+* MODIFIERS FOR READ:
+UFBAVHOLD                  EQU X'01'       (HOLD BLOCK EXCLUSIVELY)
+UFBAVREL                   EQU X'04'       (RELATIVE READ)
+UFBAVKEYED                 EQU X'04'       (KEYED READ)
+UFBAVNODATA                EQU X'08'       (DO NOT MOVE DATA TO WORK
+*                                           AREA ON READ)
+UFBAVBATCH                 EQU X'02'       (BATCH READ)
+UFBAVRESET                 EQU X'10'       (RESET READ)
+UFBAVMODAGAIN              EQU X'20'       (READ AGAIN)
+* MODIFIER FOR WRITE/DELETE (RELATIVE DISK ONLY)
```

```
+UFBAVEOF                   EQU X'02'       (WRITE OR DELETE EOF)
+* MODIFIERS FOR READ OR REWRITE (WORKSTATION ONLY):
+UFBAVTABS                  EQU X'10'       (READ OR REWRITE TABS - WS)
+* MODIFIERS FOR READ (WORKSTATION ONLY):
+UFBAVMOD                   EQU X'02'       (READ MODIFIABLE - WS)
+UFBAVALTR                  EQU X'40'       (READ ALTERED - WS)
+* MODIFIERS FOR REWRITE (WORDSTATION ONLY):
+UFBAVSELW                  EQU X'40'       (REWRITE SELECTED - WS)
+* MODIFIERS FOR START(DISK ONLY:(INDEXED:(INPUT, IO, SHARED MODES))):
+*                                (CONSECUTIVE:(SHARED MODE)):
+UFBAVEQ                    EQU X'01'       (EQUAL TO)
+* MODIFIERS FOR START(DISK ONLY:(INDEXED:(INPUT, IO, SHARED MODES))):
+UFBAVGT                    EQU X'02'       (GREATER THAN)
+UFBAVGE                    EQU X'03'       (GREATER THAN OR EQUAL TO)
+* MODIFIERS FOR START(DISK ONLY:(RELATIVE:(INPUT, IO))):
+UFBAVLT                    EQU X'10'       (LESS THAN)
+UFBAVLE                    EQU X'11'       (LESS THAN OR EQUAL TO)
+* MODIFIER FOR START (SHARED MODE; IGNORED FOR INPUT & IO MODES):
+UFBAVHFILE                 EQU X'80'       (HOLD FILE)
+UFBAVRLS                   EQU X'20'       (RELEASE HELD FILE)
+UFBAVRANGE                 EQU X'04'       (HOLD REQUEST FOR A RANGE)
+UFBAVRETRIEVAL             EQU X'40'       (HOLD CLASS IS RETRIEVAL)
+UFBAVLIST                  EQU X'10'       (LIST OPTION)
+* MODIFIERS FOR START (CONSECUTIVE OUTPUT & EXTEND MODES ONLY):
+UFBAVINPUT                 EQU X'04'       (CHANGE TO TEMPORARY IO MODE
+UFBAVOUTPUT                EQU X'08'       (CHANGE TO OUTPUT MODE)
+UFBAVEXTEND                EQU X'20'       (CHANGE TO EXTEND MODE)
+* MODIFIERS FOR START(CONSECUTIVE:(INPUT, I/O, SHARED MODES ONLY)):
+UFBAVBEGIN                 EQU X'10'       (BEGINNING OF FILE)
+UFBAVSKIP                  EQU X'40'       (FROM CURRENT RECORD
+*                                            USING SIGNED WORD
+*                                            ADDRESSED BY KEYAREA)
+* MODIFIERS FOR START(CONSECUTIVE:(I/O, SHARED MODES ONLY)):
+UFBAVEND                   EQU X'02'       RESET END OF FILE
+* MODIFIERS FOR START (PHYSICAL ACCESS METHOD ONLY):
+UFBAVCMD                   EQU X'80'       (***VAGUE NOTE***)
+UFBAVWAIT                  EQU X'40'       (WAIT FOR I/O COMPLETION)
+UFBAVWAITS                 EQU X'41'       WAIT FOR TC I/O COMPLETION
+*                                           ON THIS DEVICE ONLY
+UFBAVWAITM                 EQU X'42'       WAIT FOR TC I/O COMPLETION
+*                                           ON ALL DEVICES OPENED BY
+*                                           THIS PROGRAM
+UFBAVWAITA                 EQU X'43'       WAIT FOR TC I/O COMPLETIONS
+*                                           AND TC UNSOLICIT INTERRUPTS
+UFBAVHALTIO                EQU X'20'       HALT TC IO OPERATION
+* MODIFIERS FOR START (WORKSTATION ONLY):
+UFBAVATTNT                 EQU X'10'       (TEST FOR ATTENTIONS RECEIVE
+* *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
```

```
+                   ORG UFBAVECT
+UFBAVREAD                  DS  A           ..FOR READ
+UFBAVWRITE                 DS  A           ..FOR WRITE
+UFBAVREWRITE               DS  A           ..FOR REWRITE
+UFBAVDELETE                DS  A           ..FOR DELETE
+UFBAVSTART                 DS  A           ..FOR START
+                           ORG UFBAVWRITE
+UFBAFLAGSD              DS  BL1 RUNTIME FLAGS FOR DISK PROCESSING
+UFBAFLAGSDNEWAXD        EQU X'80' ALT-INX FILE IS NEW FORMAT
+UFBAFLAGSDCONVPR        EQU X'40' AK/PK CONVERTED TO PSEUDO-REC
+UFBAFLAGSDMULTIO        EQU X'20' PSB WRITTEN WITH MULTIO FLAG ON
+UFBAFLAGSDXCASE         EQU X'10' 3-WAY BLOCK SPLIT INDICATOR
+UFBAFLAGSDIODONE        EQU X'08' LAST ALTERNATE INDEX PROCESSED
+*                                   (USED ONLY IF DFLAGSMULTIO ON)
+UFBAFLAGSDPSBALL        EQU X'04' PSB BUFFER TEMPORARILY ALLOCATED
+UFBAFLAGSDINIT          EQU X'02' PSB BUFFER INITIALIZED
+UFBAFLAGSDCHECK         EQU X'01' CHECK OFB ISSUED FOR THIS USER
+                        DS  AL3      RESET ASSEMBLY COUNTER
+                        DS  3A       . . .
+* 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).
+UFBAERRAD                  DS  A           I!O UNUSUAL CONDITION USER
+*                                   ROUTINE ENTRY POINT, OR ZERO
+UFBAEODAD                  DS  A           END OF DATA AND INVALID KEY
+*                                   USER ROUTINE
+*                                   ENTRY POINT, OR ZERO.
+UFBARECAREA                DS  A           ADDRESS IN USER-MODIFIABLE S
+*                                   OF RECORD WORK AREA
+UFBAKEYAREA                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)
+UFBAFS1                    DS  CL1         FILE STATUS BYTE 1 FOR DMS
+UFBAFS1SUCCESS             EQU X'30'       SUCCESSFUL COMPLETION
+UFBAFS1ATEND               EQU X'31'       AT END
+UFBAFS1INVKEY              EQU X'32'       INVALID KEY OR RECORD NO.
+UFBAFS1IOERR               EQU X'33'       PERMANENT I/O ERROR
+UFBAFS1CANCEL              EQU X'36'       CANCEL CODE STORED
+*                          FOR UFBF1NOMSG (OPEN,DMS,CLOSE); UFBFS2=C'0'
+*                          MSGID AT UFBVREAD FOR O/C; NO MSGID IF DMS
+UFBAFS1TIME                EQU X'37'       TIME-OUT CONDITION ON
+*                                          SHARED MODE RESOURCE WAIT
+UFBAFS1SHARE               EQU X'38'       FS FOR SHARER CONDITION
+*                                          RESOURCE WAIT
```

```
+UFBAFS1OTHER               EQU X'39'       OTHER CONDITIONS
+**
+UFBAFS2                    DS  CL1         FILE STATUS BYTE 2 FOR DMS
+*
+UFBAFS2NOINFO              EQU X'30'       NO FURTHER INFO
+**
+* THE FOLLOWING UFBFS2 VALUES ARE SET WITH UFBFS1INVKEY (X'32')
+**
+UFBAFS2SEQERR              EQU X'31'       SEQUENCE ERROR
+UFBAFS2DUPKEY              EQU X'32'       DUPLICATE KEY
+UFBAFS2NOREC               EQU X'33'       NO RECORD FOUND
+UFBAFS2BYVIOL              EQU X'34'       BOUNDARY VIOLATION
+**
+* UFBFS2BDYVIOL IS ALSO USED WITH UFBFS1IOERR (FS = C'34')
+**
+* THE FOLLOWING UFBFS2 VALUES ARE SET WITH UFBS1SHARE (X'38')
+**
+UFBAFS2ACC                 EQU X'35'       UPDATE ACCESS DENIED FOR
+*                                           USER WITH READ-ONLY RIGHTS
+*                                           IN SHARED MODE
+UFBAFS2RESERR              EQU X'36'       RESOURCE CONTROL ERROR
+UFBAFS2DEADLOCK            EQU X'37'       DEADLOCK
+**
+* THE FOLLOWING UFBFS2 VALUES ARE SET WITH UFBFS1OTHER (X'39')
+**
+UFBAFS2ROLLBK              EQU X'33'       CURRENCY LOST DURING
+*                                   ROLLBACK
+UFBAFS2INVFUN              EQU X'35'       INVALID FUNCTION OR
+*                                   FUNCTION SEQUENCE
+UFBAFS2INVCMD              EQU X'36'       INVALID COMMAND (ALIGNMENT
+*                                   OR ADDRESS ERROR FOR DIRECT 1/O)
+UFBAFS2INVLTH              EQU X'37'       INVALID LENGTH
+UFBAFS2MASK                EQU X'38'       INVALID ACCESS MASK
+*                                           (ALTERNATE INDEXED FILES)
+UFBAFS2TRLERR              EQU X'38'       TRAILER COUNT NOT EQUAL
+*                                   TO BLOCKS READ (SET BY SVC
+*                                   CLOSE ONLY)
+UFBAFS2FMTERR              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.
+UFBAFS2XFILE               EQU X'80'       DUPLICATE FILE OR
+*                                           FILE NOT FOUND
```

```
+UFBAFS2XLIB                EQU X'40'       LIBRARY NOT FOUND
+UFBAFS2XVOL                EQU X'20'       VOLUME NOT MOUNTED
+UFBAFS2XSPACE              EQU X'10'       NO SPACE ON VOLUME
+UFBAFS2XVTOC               EQU X'08'       NO VTOC SPACE ON VOLUME
+UFBAFS2XTAPELD             EQU X'08'       WRONG TAPE LABEL/DENSITY
+UFBAFS2XPOS                EQU X'04'       POSSESSION CONFLICT
+UFBAFS2XPROT               EQU X'02'       PROTECTION CLASS VIOLATION
+UFBAFS2XFORMAT             EQU X'01'       OPEN FORMAT ERROR - ERROR
+*                                           CLASS DESCRIBED IN UFBXCODE
+UFBAAMEND                  EQU *
+UFBAAMLENGTH               EQU (UFBAAMEND-UFBABEGIN)
+* ***********************************************************
+* 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).
+* ***********************************************************
+UFBABLKSIZE                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)
+UFBARECSIZE                DS  H           LOGICAL RECORD SIZE
+*                                  (MUST BE SUPPLIED BEFORE OPEN FOR
+*                                  OUTPUT OPEN MODE)
+*
+UFBAFORG                   DS  BL1         FILE ORGANIZATION
+UFBAFORGCONSEC             EQU X'01'       CONSECUTIVE
+UFBAFORGINDEXED            EQU X'02'       INDEXED
+UFBAFORGWP                 EQU X'04'       WORD PROCESSING FILE
+UFBAFORGVIBM               EQU X'08'       IBM VARIABLE-LENGTH RECORDS
+UFBAFORGREL                EQU X'08'       RELATIVE
+UFBAFORGU                  EQU X'10'       UNDEFINED RECORD FORMAT
+UFBAFORGVLEN               EQU X'20'       VARIABLE-LENGTH RECORDS
+UFBAFORGPRINT              EQU X'40'       PRINT FILE
+UFBAFORGPROG               EQU X'80'       PROGRAM FILE
+*
+UFBAF1                     DS  BL1         OPTION FLAGS
+UFBAF1NOGET                EQU X'80'       USE GETPARM = TYPE RD
+UFBAF1NODISP               EQU X'40'       USE GETPARM = TYPE ID
+* UFBF1NOGET AMD UFBF1NODISP USED BY SVC OPEN ONLY; NOT RESET BY DMS
+UFBAF1PAM                  EQU X'20'       PHYSICAL ACCESS METHOD
+UFBAF1BAM                  EQU X'10'       BLOCK ACCESS METHOD
```

```
+UFBAF1PREVO                EQU X'08'       THIS UFB PREVIOUSLY OPENED
+UFBAF1WORK                 EQU X'04'       SCRATCH THIS WORK FILE ON
+*                                          CLOSE IF SET & FILE HAS A
+*                                          TEMPORARY NAME
+UFBAF1POOL                 EQU X'02'       BUFFER POOLING FOR RAM
+*                                          (UFBBUFSTART MUST CONTAIN
+*                                          BCT ADDRESS AT OPEN TIME)
+UFBAF1OPEN                 EQU X'01'       THIS UFB OPEN IF SET
+UFBAF2                     DS  BL1         OPEN MODE FLAGS
+UFBAF2DML                  EQU X'80'       DML in progress
+UFBAF2OUT                  EQU X'40'       TO OPEN FOR OUTPUT MODE
+UFBAF2IN                   EQU X'20'       TO OPEN FOR INPUT MODE
+UFBAF2IO                   EQU X'10'       TO OPEN FOR IO MODE
+UFBAF2EXTEND               EQU X'08'       TO OPEN FOR EXTEND MODE
+UFBAF2SHARED               EQU X'04'       TO OPEN FOR SHARED MODE
+UFBAF2DALT                 EQU X'02'       DELETIONS IN PROGRESS
+*                                           ON ALT-INDEX FILE
+UFBAF2SPCL                 EQU X'02'      TO OPEN FOR SPECIAL IO
+*
+UFBAF2PLOG                 EQU X'01'       FILE PROLOGUE PRESENT
+*
+UFBADEVCLASS               DS  BL1         DEVICE CLASS (REQUIRED
+*                                          BY 'OPEN')
+UFBADEVCLASSWS             EQU X'01'       WORKSTATION
+UFBADEVCLASSTAPE           EQU X'02'       MAGNETIC TAPE
+UFBADEVCLASSDISK           EQU X'03'       DISK
+UFBADEVCLASSPRT            EQU X'04'       PRINTER
+UFBADEVCLASSTC             EQU X'05'       TC DEVICE
+UFBADEVCLASSVC             EQU X'06'       VOICE DEVICE
+UFBADEVCLASSDUMM           EQU X'FF'       DUMMY FILE
+UFBAFLAGS                  DS  BL1         FILE ATTRIBUTE FLAGS
+UFBAFLAGSUPDAT             EQU X'80'       FILE HAS BEEN CLOSED
+UFBAFLAGSCOMP              EQU X'40'       DATA RECORDS IN COMPRESSED
+*                                          FORMAT
+* ******* UFBFLAGSRECOV - RECOVERY=YES FOR BIT = ZERO ***********
+UFBAFLAGSRECOV             EQU X'20'       USE PREFORMAT AND RECOVERY
+*                                  PROCEDURES IF ZERO (INDEXED ONLY)
+UFBAFLAGSALTX              EQU X'10'       ALTERNATE INDICES IN FILE
+UFBAFLAGSLOG               EQU X'08'       CONSEC LOG FILE FLAG
+UFBAFLAGSALTP              EQU X'08'       ALTERNATE-TREE PROCESS FLAG
+UFBAFLAGSPART              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.
+UFBAFLAGSXLCLS             EQU X'02'       SHARED FILE EXCLUSIVE
+*                                          LOCK ON CLOSE FLAG
+UFBAFLAGSPRIV              EQU X'01'       PROGRAM FILE CARRIES
+*                                          ADDITIONAL ACCESS PRIVILIGES
```

```
+UFBADEVADDR           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.)
+UFBAF3                DS  0BL1 (* NAME KEPT FOR COMPATIBILITY *)
+UFBASLOTSIZE          DS  H   RELATIVE FILE SLOT SIZE
+                      ORG UFBAF3
+UFBAPRTCLASS          DS  CL1         PRINT CLASS (A-Z)
+*
+UFBAFORMNO            DS  HL1         PRINTER FORM NUMBER (BINARY)
+UFBAPRNAME            DS  CL8         PARAMETER REFERENCE NAME
+*                             (MUST ALWAYS BE SUPPLIED HERE
+*                             FOR 'OPEN')
+UFBAVOLSER            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)
+UFBADIRNAME           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)
+UFBAFILENAME          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)
+UFBAFPCLASS           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
+UFBACREATOR           DS  CL3         FILE CREATOR FOR NEW OR
+*                             EXISTING DISK FILES
+UFBAALTCNT            DS  0BL1        COUNT OF ALTERNATE INDICES
+*                             IN FILE AFTER SVC OPEN
```

```
+UFBAALTPTR                DS  A           POINTER TO AXD1-AREA FOR DMS
+*                                 PROCESSING (ALL REFERENCE TO THE
+*                                 AXD1-AREA MUST USE UFBALTPTR)
+*
+* FOR CONSEC FILES, THE ALTPTR FIELD HOLDS LOGICAL RECORD COUNT
+                          ORG UFBAALTPTR
+UFBALOGRECCNT             DS  F LOGICAL RECORD COUNT FOR START END
+* FOR RELATIVE FILES, THE ALTPTR FIELD HOLDS CURRENCY INFORMATION
+                          ORG UFBAALTPTR
+UFBARELPOS                DS  F RELATIVE FILE LOGICAL CURRENCY PTR
+* FOR DEVICES OTHER THAN DISK, THE ALTCNT FIELD IS FOR MICROCODE TYPE
+                          ORG UFBAALTCNT
+UFBAMCTYPE                DS  XL1         DEVICE TYPE
+UFBAMCTYPE2780            EQU X'01'       2780 BATCH TC
+UFBAMCTYPE3780            EQU X'02'       3780 BATCH TC
+UFBAMCTYPETCD             EQU X'03'       TC DIAGNOSTICS
+*
+* FOR TC2780, TC3780 FILES, THE ALTPTR FIELD IS USED FOR THE TC
+* BATCH STREAM OPTIONS
+UFBATCDATAOPT             DS  BL1         TC STREAM DATA OPTION
+UFBATCXMITOPT             DS  BL1         TC STREAM TRANSMIT/RECEIVE
+*                                         OPTION
+UFBATCMAXRECSZ            DS  XL1         TC STREAM MAXIMUM RECSIZE
+*                                         MINUS 1
+* FOR WORD PROCESSING WORKSTATIONS, THE ALTPTR FIELD IS USED FOR
+* EXTENDED WS-ATTENTION INFORMATION
+                          ORG UFBAALTPTR+1
+UFBAWPAID                 DS  XL3         EXTEND WS-ATTN INFORMATION
+**
+UFBAF4                    DS  BL1         ADDITIONAL DEVICE-DEPENDENT
+*                                         FLAGS
+UFBAF4NOVTOC              EQU X'80'       UNSTRUCTURED DISKETTE
+UFBAF4RLSE                EQU X'40'       RELEASE UNUSED SPACE
+*                                         ON CLOSE
+UFBAF4BLKAL               EQU X'20'       ALLOCATE SPACE FOR NEW
+*                                         DISK FILE IN BLOCKS,
+*                                         FROM UFBNBLKS
+UFBAF4VERIFY              EQU X'10'       VERIFY OPTION ON ALL
+*                                         DISK WRITES
+UFBAF4NOMSG               EQU X'08'       NO RESPECIFY OR CANCEL
+*                                         MESSAGE FOR SVC OPEN
+*                                         ALSO NO CANCEL ON CLOSE; NO
+*                                         ACK/CANCEL FOR DMS.
+UFBAF4NOACK               EQU X'04'       NO EXCEPTIONAL CONDITION
+*                                         ACKNOWLEDGMENT MESSAGES
+*                                         FOR DMS FUNCTIONS
+UFBAF4PMSG                EQU X'02'       FOR INTERNAL USE BY DMS -
+*                                         CLOSE SENDS MESSAGE TO
+*                                         UNSPOOLER IF SET
```

```
+UFBAF4ALLOWT               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.
+UFBANRECS                  DS  FL3         NUMBER OF DATA RECORDS IN
+*                                          FILE (EXAMINED BY 'OPEN' FOR
+*                                          OUTPUT OPEN MODE ONLY.
+*                                          EXCLUDES INDEX RECORDS, ETC)
+UFBANRECSUPDAT             EQU X'80'       HI BIT SET IN NRECS HI
+*                                          BYTE (RETURNED BY LOCK)
+*                                          IF ON IN OFB AT LOCK TIME
+UFBALRECSAVE               DS  H           RECSIZE SAVED HERE
+*                                          BY OPEN (BAM)
+UFBARETPD                  DS  H           RETENTION PERIOD IN DAYS
+*                                          (MAXIMUM 999)
+UFBALOCEND                 EQU *
+UFBALOCLENGTH              EQU (UFBALOCEND-UFBABEGIN)
+* ***********************************************************
+* DATA MANAGEMENT SYSTEM SECTION
+* ***********************************************************
+UFBABCB1                   DS  BL16        BUFFER CONTROL BLOCK
+*                                  (CORRESPONDS TO SVC XIO PARAMETER
+*                                  LIST)
+                           ORG UFBABCB1
+UFBAXIOFLAGS               DS  0BL1        FLAG BYTE FOR SVC XIO
+UFBAXIOFLAGSRLS            EQU X'80'       RELEASE BUFFER AFTER WRITE
+UFBAOFB                    DS  A           OFB ADDRESS
+UFBABUFCMD                 DS  0BL1        COMMAND BYTE FOR OPERATION
+UFBABUFADR                 DS  A           BUFFER MEMORY ADDRESS
+*                                          (BLOCK ADDRESS WITHIN
+*                                          BUFFER IF BUFFER LARGER
+*                                          THAN 2K)
+UFBABUFDATAL               DS  H           LENGTH IN BYTES FOR
+*                                 OPERATION
+UFBABUFOFFSET              DS  H           OFFSET OF NEXT RECORD
+*                                 IN BUFFER
+UFBABUFBLOCK               DS  FL3         (STARTING) BLOCK WITHIN
+*                                 FILE OF BUFFERED DATA
+UFBABCBFLAGS               DS  BL1         FLAGS
+UFBABCBFLAGSLOD            EQU X'01'       BUFFER CONTENTS VALID
+UFBABCBFLAGSTOR            EQU X'02'       BUFFER TO BE REWRITTEN
+UFBABCBFLAGSIO             EQU X'04'       BUFFER I/O IN PROGRESS
+UFBABCBFLAGSPROT           EQU X'10'       BUFFER IN PROTECTED MEMORY
+UFBABCBFLAGSEOB            EQU X'20'       END OF BLOCK REACHED
+UFBABCBFLAGSEOF            EQU X'40'       EOF BLOCK IN BUFFER
+**
```

```
+* THE FOLLOWING FIELDS ARE USED FOR THE TIME-OUT OPTION IN SHARED
+* MODE ONLY.
+                           ORG UFBABUFDATAL
+UFBATIMEEXIT               DS  A           EXIT ADDRESS FOR TIME-OUT
+*                                          RETURN (0 = NO TIME-OUT)
+UFBAHOLDID                 DS  CL3         INITIALS OF HOLDER OF
+*                                          RESOURCE
+UFBATIME                   DS  XL1         WAIT TIME IN SECOND
+*                                          (0 = NO WAIT)
+**
+* THE FOLLOWING FIELDS ARE USED TO RETURN STATUS INFORMATION FROM THE
+* SHARER WHEN USER'S OPEN OF A SHARED FILE FAILS WITH FILE STATUS '60'
+* AND AN OPEN ERROR CODE OF 'E029'.
+                           ORG UFBABUFDATAL
+UFBASHROPNCODE             DS  CL4      SHARER'S OPEN ERROR MSG #
+UFBASHROPNRCSZ             DS  XL2      TRUE FILE RECORD SIZE
+UFBASHROPNFORG             DS  X        TRUE FILE ORGANIZATION BYTE
+*                                           (AS PER UFBFORG)
+UFBASHROPNSPARE            DS  X        UNUSED
+**
+UFBABUFSIZE                DS  H           BUFFER SIZE
+UFBACHKSIZE                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.
+UFBARES3                   DS  BL3         RESERVED FOR INTERNAL DMS
+         ORG   UFBARES3
+UFBAXDATE                  DS  BL3         EXPIRATION DATE (EXIST FILE)
+         ORG   UFBARES3
+UFBAOUTRECS                DS  FL3         NUMBER OF RECORDS REQUESTED
+         ORG   UFBARES3
+UFBASPB                   DS  AL3         ID OF CURRENT OPEN FOR
+*                                          MULTIPLE SHARED OPENS
+UFBANBLKS                  DS  FL3         NUMBER OF 2048-BYTE BLOCKS
+*                                          IN THE FILE
+         ORG   UFBANBLKS
+UFBADMSGID                 DS  BL3        STORED MSG-ID(DMS NOMSG EXIT)
+UFBAMAXTFR                 DS  H           MAXIMUM DATA TRANSFER IN
+*                                          BYTES FOR DISK (SET BY OPEN)
+         ORG   UFBAMAXTFR
+UFBARES1                   DS  BL1         FUTURE SPARE BYTE
+UFBAOPFLAGS                DS  BL1         INTERNAL OPEN FLAGS
+UFBAOPFLAGSPFA             EQU X'80'      PRINT-FILE ASSIGNMENT TO DISK
+UFBAOPFLAGSPFS             EQU X'40'      PF - USER SUPPLIED FILE NAME
+UFBAOPFLAGSWKA             EQU X'20'      WORK-FILE ASSIGNMENT BY OPEN
+UFBAOPFLAGSPVS             EQU X'10'      PF - USER SUPPLIED VOLUME
+UFBAOPFLAGSSCAN            EQU X'08'      IN SCAN BIT (WORK/SPOOL)
+**
```

```
+UFBALF                     DS  BL1         LAST FUNCTION PERFORMED
+UFBALFOPEN                 EQU X'00'       OPEN
+UFBALFREAD                 EQU X'04'       READ
+UFBALFWRITE                EQU X'08'       WRITE
+UFBALFREWRITE              EQU X'0C'       REWRITE
+UFBALFDELETE               EQU X'10'       DELETE
+UFBALFSTART                EQU X'14'       START
+UFBALFCLOSE                EQU X'18'       CLOSE
+UFBALFNOOP                 EQU X'50'       Noop(shared files)
+UFBALFMOD                  DS  BL1         LAST FUNCTION MODIFIER
+*                                   (DOESN'T CHANGE ON 'REWRITE')
+*                                   (SEE UFBV ABOVE)
+         ORG   UFBALFMOD
+UFBAXCODE                  DS  BL1         EXTENDED OPEN EXIT CODE
+* UFBXCODE VALUES 1-8 SET FOR POSSESSION CONFLICT
+UFBAXCODENOINFO            EQU X'00'       NO FURTHER INFORMATION
+UFBAXCODEUSE               EQU X'01'       DEVICE IN USE
+UFBAXCODEDET               EQU X'02'       DEVICE DETACHED
+UFBAXCODEVOLX              EQU X'03'       VOLUME EXCLUSIVE
+UFBAXCODEPOSS              EQU X'04'       FILE POSSESSION CONFLICT
+UFBAXCODEPAGE              EQU X'05'       PAGING FILE - SYSTEM ONLY
+UFBAXCODEIMAG              EQU X'06'       IMAGE FILE (INPUT MODE ONLY)
+UFBAXCODEAOPEN             EQU X'07'       ALREADY OPEN - THIS USER
+UFBAXCODEAUSE              EQU X'08'       ALREADY IN USE - THIS USER
+*
+* UFBXCODE VALUES X'10' - X'1F' SET FOR OPEN FORMAT ERROR
+UFBAXCODETRACK             EQU X'10'       PROGRAM REQUIRES 7 TRACK
+*                                           TAPE WHILE DRIVE IS 9 TRACK
+*                                           OR VICE VERSA
+UFBAXCODEDNPRT          EQU X'11'          UFB FORG=PRINT, WHILE
+*                                           FDR FORG NOT= PRINT
+UFBAXCODEDNPRG          EQU X'12'          UFB FORG=PROG, WHILE
+*                                           FDR FORG NOT= PROG
+UFBAXCODEDNCSC          EQU X'13'          UFB FORG=CONSEC, WHILE
+*                                           FDR FORG NOT= CONSEC
+UFBAXCODEDNWP           EQU X'14'          UFB FORG=WP,  WHILE
+*                                           FDR FORG NOT= WP
+UFBAXCODEDNINX          EQU X'15'          UFB FORG=INDEXED, WHILE
+*                                           FDR FORG NOT= INDEXED
+UFBAXCODEDFGR           EQU X'16'          UFB FORG NEITHER CONSEC
+*                                           NOR INDEXED---ERROR
+UFBAXCODENREL           EQU X'17'          UFB FORG=REL, WHILE
+*                                           FDR FORG NOT= REL
+UFBAXCODENSIO           EQU X'18'          UFB FORG=ANY,MODE=SHARED
+*                                           WHILE FDR FORG NOT=INX
+UFBAEREC                   DS  H           LAST RECORD NUMBER WITHIN
+*                                   LAST BLOCK
+UFBAVERSION                DS  HL1         UFB VERSION NUMBER *******
+UFBAEBLK                   DS  FL3         LAST BLOCK NO. WITHIN FILE
+*                                   FROM 0
```

```
+UFBABUFSTART               DS  A           BUFFER MEMORY ADDRESS;
+*                                          BUFFER CONTROL TABLE
+*                                          ADDRESS BEFORE 'OPEN'
+*                                          IF BUFFER POOLING
+*                                          SPECIFIED (UFBF1POOL SET)
+UFBARDLTH                  DS  H           LENGTH IN BYTES OF
+*                                          DATA IN BUFFER
+UFBAPRTCOPIES              DS  H           NUMBER OF PRINT COPIES
+*                                          (FOR PRINTER FILES ONLY)
+                           ORG UFBAPRTCOPIES
+UFBAWPBLKSIZE              DS  X           WORD PROCESSING FILE CONTROL
+UFBAWPBLS                  DS  X           FIELDS, WP FILES BLKSIZE
+*                                          AND BYTES IN LAST SECTOR
+                           ORG UFBABUFSTART
+UFBAPTRB                   DS  FL4         FIRST BLOCK IN INDEX
+*                                  AREA OF PRIMARY EXTENT
+*                                  (INDEXED FILES)
+UFBAPTRC                   DS  FL4         LAST BLOCK IN INDEX AREA
+*                                  OF PRIMARY EXTENT
+*                                  (INDEXED FILES)
+                           ORG UFBAPTRC
+UFBASFBVERSION             DS  FL4 LOCAL COPY OF SFB FILE VERSION #
+*
+UFBADMSEND                 EQU *
+UFBADMSLENGTH              EQU (UFBADMSEND-UFBABEGIN)
+* **********************************************************
+* END OF UFB FOR ALL FILES/DEVICES EXCEPT TAPE FILES, INDEXED DISK
+* FILES, and DMS/TX 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.
+* **********************************************************
+UFBAKEYPOS                 DS  H           KEY POSITION IN LOGICAL RECO
+UFBAKEYSIZE                DS  HL1         KEY SIZE IN BYTES
+UFBAGKSIZE                 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
+UFBAHXBLK                  DS  FL3         HIGHEST-LEVEL INDEX BLOCK
+*                                  ADDRESS FOR KEYED ACCESS
+UFBADABLK                  DS  FL3         FIRST DATA BLOCK ADDRESS
+UFBAPKI                    DS  H           INDEX ITEMS PER BLOCK
+*                                  FOR OUTPUT MODE
```

```
+UFBAPTRD                   DS  FL4         FIRST BLOCK BEYOND
+*                                  PRIMARY EXTENT
+*                                  (INDEXED FILES)
+UFBAPTRI                   DS  F           NEXT AVAILABLE INDEX
+*                                  BLOCK WITHIN PRIMARY EXTENT
+*                                  INDEX AREA
+UFBAPTRN                   DS  F           NEXT AVAILABLE INDEX
+*                                  OR DATA BLOCK IN A SECONDARY
+*                                  EXTENT (INITIALLY ZERO)
+*
+               ORG UFBAHXBLK
+UFBASHRAXD1                DS  XL20 partial AXD1 area for shared
+*                                    alternate indexed files
+*
+UFBABCBIOUT                DS  BL16         BCB FOR INDEX CREATION,
+*                                    OUTPUT MODE
+*
+*        DMS/TX Before Image control area for shared indexed files
+*        (internal system use only)
+               ORG UFBABCBIOUT
+UFBABIRECAREA              DS  A    Before Image Recarea Address
+UFBABIRECSIZE              DS  H    Before Image Record Size
+UFBABIAXD1MASK             DS  BL2  Before Image Record AXD1 Mask
+*
+                           DS  8X   RESET ASSEMBLY COUNTER
+UFBAPKD                    DS  H            RECORDS PER BLOCK FOR
+*                                    OUTPUT MODE
+UFBASPAREINX               DS  XL2          (RESERVED)
+UFBAINXDISKEND             EQU *
+UFBAINXDISKLGTH           EQU (UFBAINXDISKEND-UFBABEGIN)
+* ***********************************************************
+* DMS/TX DISK FILE EXTENSION SECTION:
+*
+*  Existence of this extension section is determined by
+*        UFBVERSION = 2 or greater and UFBDEVCLASSDISK set
+*
+*  Input fields to the Open SVC are:
+*      UFBDXOM      - Open Modifiers
+*      UFBDXRECBLCK - controls Recovery Block allocation in Output
+*                            mode only
+*      UFBDXSPARE   - must be zero
+*
+*  All other fields are returned by a successful Open; input values
+*    are ignored.
+*
+* ***********************************************************
```

```
+UFBADXOM                DS  X      DMS/TX Open Modifier Flags
+*
+*    Modifiers for general use on ANY disk file. (Their use is NOT
+*    restricted to files under DMS/TX).
+*
+UFBADXOMNOMODVOL        EQU X'80'    No modification of Volume
+*                                     in Open getparms
+UFBADXOMNOMODLIB        EQU X'40'    No modification of Library
+*                                     in Open getparms.
+*                                Open exit for xlib must be set
+*                                     (except output mode)
+UFBADXOMNOMODFIL        EQU X'20'    No modification of Filename
+*                                     in Open getparms.
+*                                Open exit for xfile must be set.
+*
+UFBADXOMCKACCESS        EQU X'10'   Restrict user access rights
+*                                    to logon privileges (ignore
+*                                    special program privileges)
+*
+UFBADXOMNOACK           EQU X'08'   suppress acknowledge
+*                                    getparms in OPEN
+*
+*    Modifiers for system use only for DMS/TX files opened in non-
+*    shared modes.
+*    Warning: Improper use can compromise the integrity of a file.
+*
+UFBADXOMREORGKEY        EQU X'04'    if file requires reorg, set
+*                                     UFBDXREORGLF, UFBKEYAREA to
+*                                     incomplete function values
+UFBADXOMNOREC           EQU X'02'    No Recovery
+UFBADXOMNOCHK           EQU X'01'    No Check for file softcrash
+*                                     or reorganization required
+*
+UFBADXRECBLK            DS  C      Recovery Blocks flag
+*   Output mode: set to RECBLKALLO to allocate Recovery Blocks
+*                set to RECBLKNO   to not allocate Recovery Blocks
+*   Value is returned for all other modes.
+UFBADXRECBLKNO          EQU C'N'    No Recovery Blocks
+UFBADXRECBLKALLO        EQU C'A'    Recovery Blocks Allocated
+*                                   but not used
+UFBADXRECBLKUSED        EQU C'U'    Recovery Blocks allocated &
+*                                   used (file is under DMS/TX)
+*
+UFBADXDBNAME            DS  CL6    Database Name
+*
+UFBADXFV#               DS  0XL12  File version #
+UFBADXFV#SEQ#           DS  F       sequence #
+UFBADXFV#DT             DS  PL8     date/time stamp
+*
```

```
+UFBADXRECO             DS  C      Recovery option after Open    +
+                                  (usually the Database option) +
+                                  Not an input parameter.
+UFBADXRECONO           EQU C'N'     No Recovery
+UFBADXRECOSOFT         EQU C'S'     Softcrash Recovery
+*
+* ***********************************************************
+* The following fields are for internal system use only:
+*
+UFBADXLSBREC           DS  X      LSB File Recovery Option
+*
+UFBADXREORGLF          DS  X      UFBLF value from incomplete
+*                                  function (if UFBDXOMREORGKEY
+*                                  set and file requires reorg)
+*
+UFBADXCS               DS  X      Crash Status of DMS/TX files
+*                                  (input mode or DXOMNOCHK set)
+UFBADXCSSOFT           EQU X'01'    Softcrash Recovery required
+UFBADXCSREORG          EQU X'02'    Reorganization required
+*
+UFBADX#BIJS            DS  H      # BIJS accessing crashed file
+*                                  (if DXOMNOCHK set)
+*
+UFBADXFLAGS            DS  X      Extra flag bits
+UFBADXFLAGSTXON        EQU X'80'  Turn on dmstx locking
+*                                  protocol on file OPEN
+UFBADXFLAGSRDNLY       EQU X'20'  Open for shared read-only
+*
+UFBADXFLAGSNODXE       EQU X'40'  Ignore DXE's
+*
+UFBADXFLAGSRSN         EQU X'10'  Record Sequence Numbers Used
+*
+UFBADXFLAGSGE          EQU X'08'  LF WAS START (SPCL RD NXT)
+*
+UFBADXFLAGSWAIT        EQU X'04'  SFB WAIT FLAG HAS BEEN SET
+*
+UFBADXPSBBUF           DS  AL3    Address of PSB buffer
+*
+UFBADXSPARE            DS  XL2
+UFBADXMSB              DS  A      Pointer to MSB(multithreading)
+*
+UFBADXEND              EQU *
+UFBADXLGTH             EQU (UFBADXEND-UFBABEGIN)
+* ***********************************************************
+* 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.
+* ***********************************************************
```

```
+                           ORG UFBADMSEND
+UFBATSPARE1                DS  BL4         (RESERVED)
+UFBATBCB                   DS  BL16        ADDITIONAL BUFFER CONTROL
+*                                          BLOCK FOR TAPE DOUBLE
+*                                          BUFFERING
+UFBATLABELS                DS  BL1         REQUESTED LABELING (OUTPUT)
+*                                          OR LABEL TYPE ON TAPE
+*                                          (INPUT)
+UFBATLABELSNL              EQU X'01'       UNLABELLED
+UFBATLABELSANY             EQU X'02'       ANY TYPE OF LABEL
+UFBATLABELSAL              EQU X'04'       ASCII LABELS
+UFBATLABELSIL              EQU X'08'       IBM LABELS
+UFBATDEN                   DS  BL1         TAPE DENSITY
+UFBATDEN800                EQU X'01'       800 BPI
+UFBATDEN1600               EQU X'02'       1600 BPI
+UFBATDEN556                EQU X'03'       556 BPI
+UFBATDEN6250               EQU X'08'       6250 BPI
+UFBATDEN6400               EQU X'10'       6400 bpi
+*
+UFBATSEQ                   DS  H           TAPE FILE SEQUENCE NUMBER
+*                                          (SET BEFORE OR DURING
+*                                          OPEN TO REQUEST POSITIONING
+*                                          AND AVAILABLE AFTER OPEN)
+UFBATFLG                   DS  BL1         TAPE-RELATED FLAGS
+UFBATFLGALLOWNL            EQU X'80'       *** OBSOLETE ***
+UFBATFLGSWITCH             EQU X'40'       TAPE VOLUME SWITCH REOPEN
+*                                          IN PROGRESS
+UFBATFLGEODEOV             EQU X'20'       TAKE EOV1 TRAILER LABEL AS
+*                                          EOF1 LABEL
+UFBATFLG7TRACK             EQU X'10'       USE 7 TRACK TAPE DRIVE FOR
+*                                          THIS FILE
+UFBATFLGNOHDR2             EQU X'08'       NO HDR2 FILE LABEL
+UFBATVOLSEQ                DS  BL1         TAPE VOLUME SEQUENCE NUMBER
+*                                          (ORDER OF VOLUME IN A
+*                                          MULTIPLE VOLUME FILE)
+UFBATSAVEVOL               DS  CL6         VOLUME NAME OF FIRST
+*                                          VOLUME OF A MULTI-VOLUME
+*                                          FILE SAVED HERE
+UFBATPARITY                DS  BL1         TAPE PARITY (7 TRACK TAPE
+*                                          ONLY)
+UFBATPARITYODD             EQU X'01'       ODD PARITY
+UFBATPARITYEVEN            EQU X'02'       EVEN PARITY
+UFBATSPARE2                DS  BL11        (RESERVED - MUST BE 0)
+UFBATAPEEND                EQU *
+UFBATAPELGTH               EQU (UFBATAPEEND-UFBABEGIN)
+                           ORG UFBADXEND
+UFBAEND                    EQU *
+UFBALGTH                   EQU (UFBAEND-UFBABEGIN)
+         CSECT
+         USING UFBA,(R2)
```

4.2.80 UFBGEN - Generate a User File Block

Syntax

```
[label]  UFBGEN    [,ALLOWNL={NO }]          [,ALLOWTAPE={NO }]
                             {YES}                       {YES}

                   [,ALTAREA={(register)}]   [,ALTCNT={ 0  }]
                             { address  }             {0-16}
                             {    0     }

                   [,BAM={NO }]              [,BCT={(register)}]
                         {YES}                     { address  }

                   [,BLKAL={NO }]            [,BLKSIZE={integer}]
                           {YES}

                   [,BUFSIZE={integer}]      [,COMP={NO }]
                             { 2048  }              {YES}

                   [,CKACCESS={NO }]         [,DEN={556 }]
                              {YES}                {800 }
                                                   {1600}
                                                   {6400}

                   [,DEVCLASS={DISK }]       [,DEVNO=integer]
                              { PRT }
                              {  WS }
                              {MTAPE}

                   [,DMSTXSECTION={NO }]     [,DPACK={100  }]
                                  {YES}              {1-100}

                   [,EOD=EOV]                [,EODAD={(register)}]
                                                     {  address }

                   [,ERRAD={(register)}]     [,FILECLAS={ 0  }]
                           { address  }                 {#,A-Z}

                   [,FILENAME={(register)}]  [,FORG={CONSEC }]
                              { address  }          {INDEXED}
                                                    {  ANY  }

                   [,FORM={  0  }]           [,FSEQ=integer]
                          {1-255}

                   [,HEADER={PARTIAL}]       [,IPACK={100 }]
                            {FULL   }                {1-99}

                   [,KEYAREA={(register)}]   [,KPOS={(register)}]
                             {  address }           {  address }
```

```
[,KSIZE={(register)}]    [,LABEL={NL }]
        {  address }             {AL }
                                 {IL }
                                 {ANY}

[,LIBRARY={(register)}]  [,MCTYPE={2780  }]
          { address  }            {3780  }
                                  {TCDIAG}

[,MODE={  OUT }]         [,NBLKS={(register)}]
       {  IN  }                  {  address }
       {  IO  }
       {EXTEND}
       {SHARED}

[,NODISPLAY={NO }]       [,NOVTOC= {NO }]
            {YES}                  {YES}

[,NRECS={(register)}]    [,OPENNOACK={NO }]
        { address  }                 {YES}
        {    0     }

[,PAM={NO ]              [,PARITY={EVEN}]
      {YES}                       {ODD }

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

[,PRINT={NO }]           [,PROG={NO }]
        {YES}                   {YES}

[,PRNAME={(register)}]   [,PRTCLASS={A  }]
         { address  }               {B-Z}

[,RECAREA={(register)}]  [,RECBLKS={NO }]
          {  address }             {YES}

[,RECSIZE={integer}]     [,STREAM={READER }]
          { ANY    }              {PUNCH  }
                                  {PRINTER}

[,TRANSMISSION={TRANPARENT    }][,TRACK7={NO }]
               {NONTRANSPARENT}          {YES}
               {UNBLOCKED     }
               {BLOCKED       }
               {UNPADDED      }
               {COMPRESSED    }
               {PADDED        }
               {UNCOMPRESSED  }
               {EBCDIC        }
               {ASCII         }
```

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

[,VLEN={NO }]               [,VOLSER={(register)}]
       {YES}                         {  address }

[,VSEQ={integer}]
       {   1   }
```

Function

Generates a user file block (UFB) with the specified fields initialized. This macroinstruction does not produce executable code.

Parameter Definitions

ALLOWNL Allows nonlabelled tape. Default is NO.

ALLOWTAPE If set to YES, OPEN allows tape as an alternative device for disk file.

ALTAREA Address of AXD1 block, as generated by the AXDGEN macroinstruction.

ALTCNT An integer between 0 and 16. If not 0, ALTAREA parameter must be supplied. This is the number of alternate indices processable for the file.

BAM Optional request to process a disk file as if it had 2048-byte logical records, regardless of the record size recorded in its file descriptor record. Defaults to BAM=NO.

BCT Address of the buffer control table generated by BCTGEN. Must be specified if POOL=YES is specified.

BLKAL Allocate space for a new disk file in number of blocks, as specified in UFBNBLKS (see NBLKS parameter), rather than in number of logical data records.

BLKSIZE Sizes are used for existing files to verify the 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. Refer to the DMS Reference Manual for details.

COMP Specifies whether records are to be compressed or not.

CKACCESS YES restricts access to user's logon access rights. Ignores special program privileges. Only legal if DMSTXSECTION=YES is specified.

DEN Magnetic tape density, 556 for 556 BPI tape, 800 for 800 BPI tape, 1600 for BPI tape, 6400 for 6400 BPI tape.

DEVCLASS Valid options are DISK, MTAPE, WS, TC, or PRT.

DEVNO The device number option can be used for print files to request printing on a specific printer.

DMSTXSECTION Specifying YES allocates UFB with DMSTX section.

DPACK Used to specify the relative percentage of used to unused space (packing) desired for data blocks on a new indexed file. If not specified, system default values are used.

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.

EODAD Specifies the address in the user program where control is returned to in an end-of-data situation.

ERRAD Specifies, the address where control is returned to when a DMS fatal error is encountered.

FILECLAS Specifies the file protection class.

FILENAME Specifies the file name.

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.

FSEQ Tape file sequence number.

HEADER This parameter 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 cancels with an indication of an invalid label type. If HEADER=PARTIAL is specified, but a HDR2 label is found on the tape, OPEN proceeds 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.

IPACK Specifies the relative percentage of used to unused space (packing) desired for index blocks on a new indexed file. If not specified, system default values are used.

KEYAREA These parameters must be acceptable in "DC A(pn)" assembler statements. They may be written only when the UFB is generated in a data static area.

KPOS Specifies the key position.

KSIZE Specifies the key size.

LABEL Magnetic tape label types allowed: NL for no label, AL for ANSI label, IL for IBM label. None, one, or more than one can be specified.

LIBRARY Specifies the library where the file exists or is to be created.

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.

MODE Specifies the mode in which the file is to be opened, Output, Input, Input/Output, Extend or Shared.

NBLKS Only used with Output mode in PAM or BAM. Specifies how many blocks to allocate to the file.

NODISPLAY If YES is specified, then the OPEN SVC does not issue a GETPARM to the user's workstation for CANCEL messages or for respecification messages.

NOVTOC Optional file attribute for diskette only. Specify only for unstructured diskette.

NRECS The number of records. Used for Output mode only for RAM. Specifies the number of records used to calculate the number of extents necessary for the file.

OPENNOACK Used only with DMSTXSECTION=YES. Suppresses OPEN acknowledge GETPARM.

PAM Optional request for physical access method support or multiple-line printing. Defaults to PAM=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.

PLOG If YES is specified, indicates that a file prologue is present. Valid only for word processing files. This parameter is applicable only when OPEN mode is OUTPUT, and is ignored for any other OPEN mode. The default is NO.

POOL Buffer pooling requested. The BCT parameter addresses a buffer control table in the user-modifiable data area, as created by the BCTGEN macroinstruction.

PRINT Specifies that the file in question is to be a PRINT file. On input reads the file as a print file. On output creates the file in PRINT file organization.

PROG These attributes are used for existing files to limit acceptance to files of the indicated attributes.

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.

PRTCLASS Used to specify the print queue class.

RECAREA Used to specify the address of the user record area.

RECBLKS Used only for Output mode when DMSTXSECTION=YES. RECBLKS specifies whether to allocate recovery blocks.

RECSIZE The record size (or maximum record size) and block.

STREAM Sets the TC STREAM DATA option in UFBTCDATAOPT.

TRANSMISSION Sets the TC STREAM TRANSMIT/RECEIVE options in UFBTCXMITOPT.

TRACK7 If YES is specified, then 7-track tape is indicated. The default is NO. The user must specify a nonzero value for tape density in the UFB in the case of a 7-track tape.

VERIFY Request read-after-write verification on a disk file.

VLEN Specifies whether record length is variable or not.

VOLSER Specifies the name of the volume on which the file resides.

VSEQ Tape volume sequence number for multiple volume tape file.

The following UFBGEN parameters are only used for physical access method (PAM) or block access method (BAM):

- PAM=YES
- BAM=YES
- BLKALS=YES
- NBLKS=p13

The other UFBGEN parameters are for the random record access method (RAM). Table 4-2 summarizes the use of the parameters used with RAM. The legend for using Table 4-2 is as follows:

R Required for OPEN processing. Can optionally be set by
1 the program prior to OPEN.

O Optional for OPEN processing. Can also be set by the
1 program before OPEN.

R Required for DMS functions. Can be set by the program
2 before use.

O Optional for DMS functions. Can be set by the program
2 before use.

__ Underlines are used to identify default values.

Table 4-2
PARAMETER USAGE TABLE
RECORD ACCESS METHOD

COMMONLY USED PARAMETERS
FOR FILES

PARAMETER	NEW			EXISTING		
	CONSECUTIVE DISK FILES	INDEXED DISK FILES	WORK OR TEMP DISK FILES	DISK FILES	NEW PRINT FILES	WORKSTATION FILES
PRNAME	R 1-8 CHAR 1 ALPHA NUMERIC	R 1-8 CHAR 1 ALPHA- NUMERIC	R 1-8 CHAR 1 ALPHA - NUMERIC	R 1-8 CHAR 1 ALPHA - NUMERIC	R 1-8 CHAR 1 ALPHA- NUMERIC	R 1-8 CHAR 1 ALPHA- NUMERIC
DEVCLASS	R 1 DISK	R 1 DISK	R 1 DISK	R 1 DISK	R PRT 1	R WS 1
MODE	R 1 OUT	R 1 OUT	R 1 OUT	R IN, IO, 1 EXTEND SHARED	R 1 OUT	R 1 IO
FORG	R 1 CONSEC	R 1 INDEXED	R 1 CONSEC[7] INDEXED	R CONSEC, 1 INDEXED, ANY	R 1 CONSEC	R 1 CONSEC
VLEN	0 YES 1 NO	0 NO 1 YES	NO YES	0 NO 1 YES	R YES 1 NO	
COMP	0 NO 1 YES	0 NO 1 YES	NO YES	NO YES	R YES 1 NO	
PRINT	0 NO 1 YES		NO YES	0 NO 1 YES	R YES 1 NO	
PROG	0 NO 1 YES		NO YES	0 NO 1 YES		
NRECS[2]	R 1 NUMERIC	R 1 NUMERIC	R 1 NUMERIC		R 1,000 1 NUMERIC	
RECSIZE	R 1 NUMERIC	R 1 NUMERIC	R 1 NUMERIC	R ANY, 1 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	
FORM[5]					0 (0,1-255) 1	

Table 4-2
PARAMETER USAGE TABLE
RECORD ACCESS METHOD (continued)

	NEW			EXISTING		
PARAMETER	CONSECUTIVE DISK FILES	INDEXED DISK FILES	WORK OR TEMP DISK FILES	DISK FILES	NEW PRINT FILES	WORKSTATION FILES
PRTCLASS[5]					0 (A,B-Z) 1	
DEVNO					0 1 NUMERIC	
KPOS		R R NUMERIC				
KSIZE		R 1 NUMERIC				
DPACK		0 NUMERIC 1 1-100				
IPACK		0 NUMERIC 1 1-100				
NOVTOC[3]	0 NO 1 YES			0 NO 1 YES		
VERIFY[4]	0 NO 1 YES	0 NO 1 YES	0 NO 1 YES		0 NO 1 YES	
FILENAME[5]	0 1-8 CHAR[5] 1 ALPHA- NUMERIC	0 1-8 CHAR 1 ALPHA- NUMERIC	R 1-8 CHAR 1 ALPHA- NUMERIC	0 #AAAA or 1 ##AAA SEE NOTE 1		
LIBRARY[5]	0 1-8 CHAR 1 ALPHA- NUMERIC	0 1-8 CHAR 1 ALPHA- NUMERIC		0 1-8 CHAR 1 ALPHA- NUMERIC		
VOLSER[5]	0 1-6 CHAR 1 ALPHA- NUMERIC	0 1-6 CHAR 1 ALPHA- NUMERIC		0 1-6 CHAR 1 ALPHA- NUMERIC		
FILECLAS[5]	0 (#,A-Z) 1	0 (#,A-Z) 1	R (#,A-Z) 1		R (#,A-Z) 1	

Table 4-2
PARAMETER USAGE TABLE
RECORD ACCESS METHOD (continued)

PARAMETER	NEW			EXISTING		
	CONSECUTIVE DISK FILES	INDEXED DISK FILES	WORK OR TEMP DISK FILES	DISK FILES	NEW PRINT FILES	WORKSTATION FILES
NODISPLAY[6]	0 YES 1	0 YES 1	R YES 1	0 YES 1	R 1 YES	R 1 YES
RECAREA	R ADDRESS 2 IN DATA SECTION	R ADDRESS 2 IN DATA SECTION	R ADDRESS 2 IN DATA SECTION	0 ADDRESS 1 IN DATA SECTION	R ADDRESS 2 IN DATA SECTION	R ADDRESS 2 IN DATA SECTION
KEYAREA	ADDRESS IN DATA SECTION	R ADDRESS 2 IN DATA SECTION	ADDRESS IN DATA SECTION	0 ADDRESS 2 IN DATA SECTION		ADDRESS IN DATA SECTION
ERRAD	0 INSTRUCTION 2 ADDRESS	0 INSTRUCTION 2 ADDRESS	0 INSTRUCTION 2 ADDRESS	0 INSTRUCTION 2 ADDRESS	0 INSTRUCTION 2 ADDRESS	0 INSTRUCTION 2 ADDRESS
EODAD	0 INSTRUCTION 2 ADDRESS	0 INSTRUCTION 2 ADDRESS	0 INSTRUCTION 2 ADDRESS	0 INSTRUCTION 2 ADDRESS	0 INSTRUCTION 2 ADDRESS	0 INSTRUCTION 2 ADDRESS

Notes on Table 4-2

(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 work file 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 the NOVTOC files. When NOVTOC is used, the 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 parameters represent run-time parameters ultimately determined via GETPARM. Unless required to identify WORK or TEMP files, these parameters 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 parameters must be consistent.

Example

```
         UFBGEN EODAD=CLFILES,PRNAME=INPUT,BUFSIZE=18*1024,            X
               FORG=ANY,MODE=IN,DEVCLASS=DISK
+* USER FILE BLOCK FOR PRNAME 'INPUT'

+* ACCESS METHOD SECTION
+        DS    5A(0)         DATA MANAGEMENT ROUTINES VECTOR
+        DC    A(0)          ERROR EXIT NOT SPECIFIED
+        DC    A(0)          END OF DATA AND INVALID KEY EXIT OMITTED
+        DC    A(0)          RECORD WORK AREA ADDRESS OMITTED
+        DC    A(0)          KEY AREA ADDRESS OMITTED
+        DS    2C            FILE STATUS BYTES
```

```
+* FILE LOCATION AND ATTRIBUTES SECTION
+         DC    AL2(0)        PHYSICAL BLOCK SIZE
+         DC    AL2(0)        RECORD SIZE OMITTED
+         DC    AL1(0)        FILE ORGANIZATION
+         DC    AL1(0)        FLAG BYTE (ATTRIBUTES)
+         DC    AL1(32)       OPEN MODE
+         DC    XL1'03'       DISK DEVICE CLASS
+         DC    AL1(0)        FLAGS
+         DC    XL1'FF'       DEVICE ADDRESS NOT SUPPLIED
+         DC    CL1' '        PRTCLASS OMITTED
+         DC    AL1(255)      FORM NUMBER OMITTED
+         DC    CL8'INPUT'    PRNAME
+         DC    CL6' '        VOLUME SERIAL NOT SPECIFIED
+         DC    CL8' '        LIBRARY NAME NOT SPECIFIED
+         DC    CL8' '        ACTUAL FILE NAME NOT SPECIFIED
+         DC    X'00'         FILECLASS OMITTED
+         DC    CL3' '        USER ID
+         DC    AL1(0)        ALTCNT
+         DC    AL3(0)        ALTPTR
+         DC    AL1(0)        DEVICE-DEPENDENT FLAGS
+         DC    AL3(0)        NUMBER OF DATA RECORDS
+* DATA MANAGEMENT SYSTEM SECTION
+         DC    20X'00'       (UNSET BEFORE OPEN)
+         DC    AL2(18*1024)  BUFFER SIZE
+         DC    5X'00'        (UNSET BEFORE OPEN)
+         DC    AL3(0)        NUMBER OF DATA BLOCKS
+         DC    6X'00'        (UNSET BEFORE OPEN)
+         DC    X'01'         VERSION NUMBER
+         DC    11X'00'       (UNSET BEFORE OPEN)
+*
+* INDEXED DISK FILE EXTENSION SECTION
+         DC    H'0'          KEY POSITION NOT SPECIFIED
+         DC    HL1'0'        KEY SIZE NOT SPECIFIED
+         DC    HL1'0'        KEY SIZE OVERRIDE
+         DC    FL3'0'        HXBLK
+         DC    FL3'0'        DABLK
+         DC    H'100'        INDEX BLOCK PACKING
+         DC    28BL1'0'      PTRD,PTRI,PTRN,BCBIOUT
+         DC    H'100'        DATA BLOCK PACKING
+         DC    BL2'0'        (RESERVED)

+         DC    CL6' '        VOLUME SERIAL NOT SPECIFIED
+         DC    CL8' '        LIBRARY NAME NOT SPECIFIED
+         DC    CL8' '        ACTUAL FILE NAME NOT SPECIFIED
+         DC    X'00'         FILECLASS OMITTED
+         DC    CL3' '        USER ID
+         DC    AL1(0)        ALTCNT
+         DC    AL3(0)        ALTPTR
+         DC    AL1(0)        DEVICE-DEPENDENT FLAGS
+         DC    AL3(0)        NUMBER OF DATA RECORDS
```

```
+* DATA MANAGEMENT SYSTEM SECTION
+          DC     20X'00'        (UNSET BEFORE OPEN)
+          DC     AL2(2048)      BUFFER SIZE
+          DC     5X'00'         (UNSET BEFORE OPEN)
+          DC     AL3(0)         NUMBER OF DATA BLOCKS
+          DC     6X'00'         (UNSET BEFORE OPEN)
+          DC     X'01'          VERSION NUMBER
+          DC     11X'00'        (UNSET BEFORE OPEN)
+*
+* INDEXED DISK FILE EXTENSION SECTION
+          DC     H'0'           KEY POSITION NOT SPECIFIED
+          DC     HL1'0'         KEY SIZE NOT SPECIFIED
+          DC     HL1'0'         KEY SIZE OVERRIDE
+          DC     FL3'0'         HXBLK
+          DC     FL3'0'         DABLK
+          DC     H'100'         INDEX BLOCK PACKING
+          DC     28BL1'0'       PTRD,PTRI,PTRN,BCBIOUT
+          DC     H'100'         DATA BLOCK PACKING
+          DC     BL2'0'         (RESERVED)
+          DC     BL2'0'         (RESERVED)
```

4.2.81 UNITRES - Reserve/Release Telecommunications Devices, Lines, and Peripheral Processors (SVC 51)

Syntax

```
[label]  UNITRES   {RESERVE},DEVICE={     address      }[,DIAGNOSTICS]
                   {RELEASE}        {    (register)    }
                                    {self-defining term}

                          PP={      address      }
                             {     (register)    }
                             {self-defining term}

                          IOP={     address      }
                              {    (register)    }
                              {self-defining term}
```

Function

The UNITRES macro is used to reserve and release exclusive use of non-shareable devices, peripheral processors (PPs), and I/O processors (IOPs).

The RESERVE option checks to see if the device can be acquired. In the case of a processor (either a peripheral or I/O), the RESERVE option checks to see if all the devices associated with the processor can be acquired, and if so, allocates them for exclusive use so that I/O can be initiated to the processor. A processor may be reserved if it is not already reserved by another task, if none of its associated devices are opened or reserved, or if it is already reserved by the user.

For reservation of a particular device, the RESERVE parameter is specified along with the DEVICE parameter. The DEVICE parameter value is the device number associated with the device through the SYSGEN process.

In reserving a processor, the RESERVE parameter is specified, together with the PP or IOP parameter. The value for either of these parameters is the device number of any one of the devices associated with the processor through the SYSGEN procedure.

Reservation is exclusive and remains in effect until a UNITRES RELEASE is performed. To release a device, the RELEASE option is specified with the DEVICE parameter. The value of the DEVICE parameter should be the device number specified in the UNITRES RESERVE statement that initially reserved the device.

To release a previously reserved processor, the RELEASE option is specified, together with the PP or IOP parameter. The value for the PP or IOP parameter should be the device number specified in the UNITRES RESERVE statement that initially reserved the processor.

A processor can be released only if it and all associated devices were previously reserved exclusively by the caller.

Parameter Definitions

RESERVE Allows reservation of a device or a peripheral processor.

RELEASE Releases the device or peripheral processor.

DEVICE Device number of the device to be reserved or released. This parameter can be specified as a self-defining term which is the device address, as a register in parentheses that contains the device address in its low-order byte, or as an expression that addresses a 1-byte field which contains the device address.

PP Device address of a device associated with the peripheral processor to be reserved or released. This parameter can be specified as a self-defining term which is the device address, as a register in parentheses that contains the device address in its low-order byte, or as an expression that addresses a 1-byte field which contains the device address.

IOP Device number of any one of the devices associated with the I/O processor to be reserved or released. This parameter can be specified as a self-defining term which is the device number, as a register in parentheses that contains the device number in its low-order byte, or as an expression that addresses a 1-byte field which contains the device number.

DIAGNOSTICS Used with the RESERVE DEVICE option to reserve a workstation for diagnostic purposes. Diagnostic privilges must be acquired through the Security program for this function to work at runtime. Specifying DIAGNOSTIC is not required on the RELEASE option.

Output

UNITRES returns a fullword at the top of the stack, indicating the success or failure of the RESERVE or RELEASE operation.

Return Codes

Code	Description
0	Success.
4	Invalid unit address.
8	Invalid function code.
12	Invalid unit type.
16	Reserved.
20	PP specified for nonprogrammable device.
24	PP reservation conflict.
28	Reserved.
32	Release specified for a device or PP that the caller does not own.
36	Specified device is a disk.
40	Device reservation conflict.
44	Invalid device specified for diagnostics.
48	No privileges for diagnostics.

Examples

```
 DONE     UNITRES RESERVE,DEVICE=TERM
+DONE     PUSHA   0,0                   CLEAR 4 BYTES OF STACK SPACE
+         MVI     0(15),X'01'           SET FUNCTION = "RESERVE"
+         MVI     1(15),X'01'           SET UNITTYPE = DEVICE
+         MVC     3(1,15),TERM          SET UNIT ADDRESS
+         SVC     51  (UNITRES)

 FINISH   UNITRES RELEASE,DEVICE=TERM
+FINISH   PUSHA   0,0                   CLEAR 4 BYTES OF STACK SPACE
+         MVI     0(15),X'02'           SET FUNCTION = "RELEASE"
+         MVI     1(15),X'01'           SET UNITTYPE = DEVICE
+         MVC     3(1,15),TERM          SET UNIT ADDRESS
+         SVC     51  (UNITRES)
```

```
 DSTART   UNITRES RESERVE,DEVICE=(R2),DIAGNOSTICS
+DSTART   PUSHA 0,0                 CLEAR 4 BYTES OF STACK SPACE
+         MVI   0(15),X'01'         SET FUNCTION = "RESERVE"
+         MVI   2(15),X'01'         SET FLAG = DIAGNOSTICS           02\
+         MVI   1(15),X'01'         SET UNITTYPE = DEVICE
+         STC   R2,3(15)            SET UNIT ADDRESS
+         SVC   51  (UNITRES)
          END BEGIN
```

4.2.82 UPDATFDR - Update File Descriptor Record (SVC 25)

Syntax

```
[label]  UPDATFDR  [PLIST={(register)}][,VOLUME={(register)}]
                          { address  }          { address   }
                                                { 'string'  }

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

                   [,CLOSE={YES}][,RELEASE={YES}]
                           {NO }           {NO }

                   [,RESTRICT={YES}]
                              {NO }

                   [,NRECS={(register)},EBLK={(register)},
                           { address  }      { address  }

                   EREC={(register)}][,WPBLKSIZ={(register)},
                        { address  }            { address  }

                   WPBLS={(register)}][,HXBLK={(register)},
                         { address  }         { address  }

                   DABLK={(register)},PTRD={(register)}]
                         { address  }      { address  }

                   [,EXTCOUNT={(register)},SECEXT={(register)},
                              { address  }        { address  }

                   EXTPTR={(register)}][,WTFLGS={(register)},
                          {  address }          { address  }

                   EXFLGS={(register)},RDFLGS={(register)}]
                          { address  }        { address  }

                   [,FPCLASS={(register)},CREATOR={(register)},
                             { address  }         { address  }
                             { 'string' }         { 'string' }

                   CREDATE={(register)},MODDATE={(register)}
                           { address  }         { address  }

                   EXPDATE={(register)}][,FGPRIORITY={(register)}]
                           { address  }              { address  }

                   [,BGPRIORITY={(register)}]
                                { address  }
```

Function

Updates an existing file descriptor record (FDR) in the volume table of contents (VTOC) of the specified volume. The file descriptor record of the file is modified with the data given in the argument list.

The modifiable FDR1 entries are organized into six option groups. Each option group exists in the argument list only if the corresponding bit in the option flag is set to 1. All option groups can be combined in any way as specified in the option flag. The argument list is interpreted in the group order of 1, 2, 3, 4, 5, 6.

- Group 1 updates the following items:
  - Number of data records
  - Last record's block within the file
  - Last record's number in last block for consecutive files with fixed-length record (For indexed files, number of primary index levels)

- Group 2 updates the following items for indexed files:
  - Block in file of root block of primary index
  - Block in file of starting block of available-block chain
- First data block in file

- Group 3 updates the following information:
  - Count of extents in use
  - Number of blocks in secondary extent
  - Extent list pointer

- Group 4 updates the following information:
  - Expiration date of the file
  - Modified date
  - Creation date
  - User ID of the creator

- Group 5 updates the following items for program files:
  - Access privileges
  - Write privileges
  - Read privileges
  - Extend privileges

- Group 6 updates the following items from program files:
  - Foreground priority
  - Background priority

Restrictions

Group 1 or 2, or both, can be updated if the specified file is opened in an exclusive mode (IO, OUTPUT, EXTEND) by the issuer.

The user must be privileged to update group 3. Group 3 should not be specified in conjunction with release unused space in extents. If this is done, the release option is ignored.

To update group 4 or group 5, the specified file must be closed.

If group 4 is to be updated, either the issuing program or user must be the file creator as named in (the previous value of) FDR1CREATOR, or the issuing program or user must have protection system administrator access rights to files.

If group 5 is specified and is set to all zeroes, FDR1FLAGSPRIV is set to 0. If group 5 is specified and is not all zeroes, FDR1FLAGSPRIV is set to 1.

If group 5 or 6 is to be updated, the issuing user must have system administrator access rights. Option bit 7 may be used to limit access rights to the user's logon rights (for all groups).

The area addressed by PLIST must be in the user's data section. If any parameters are supplied as character strings (and in some other cases), the user must allow for generation of a literal pool.

Parameter Definitions

PLIST An address or a register in parentheses, pointing to a user-generated parameter list to be used by the UPDATFDR SVC. If PLIST is specified, no other parameter is required, and it is assumed that the user has placed appropriate values in the PLIST for parameters omitted which would have been required.

VOLUME Indicates the volume that contains the FDR to be updated. This parameter can be specified as an address expression, a register in parentheses that points to a 6-byte field which contains the volume name, or a character string in single quotes. Required unless PLIST is specified.

LIBRARY Indicates the library that contains the file whose FDR is to be updated. This parameter can be specified as an address expression, a register in parentheses that points to an 8-byte field which contains the library name, or a character string in single quotes. Required unless PLIST specified.

FILE Indicates the file whose FDR is to be updated. This parameter can be specified as an address expression, a register in parentheses that points to an 8-byte field which contains the library name, or a character string in single quotes. Required unless PLIST is specified.

CLOSE If YES is specified, the update bit in FDR1FLAG is set. Required unless PLIST is specified. The default is NO.

RELEASE If YES, unused space in the file is released. Required unless PLIST is specified. The default is NO. Ignored if EXTCOUNT, SECEXT, or EXTPTR is specified.

RESTRICT If NO, then any current special access rights granted to the invoking program are honored. If YES, file access is restricted to the user's LOGON access rights. Required unless PLIST is specified. The default is YES.

NRECS Indicates the number of records in the file. This parameter can be specified as an address that points to a 4-byte binary number, or a register in parentheses that contains a number. Required if PLIST is not specified, and either EBLK or EREC is specified. The file must be open in Exclusive mode.

EBLK Indicates the last record's block number within the file. This parameter can be specified as an address that points to a 3-byte binary number, or a register in parentheses that contains the number. Required if PLIST is not specified, and either NRECS or EREC is specified. The file must be open in Exclusive mode.

EREC Indicates the number of the last record in the last block of the file. This parameter can be specified as an expression that points to a 2-byte binary number, or a register in parentheses that contains the number. Required if PLIST is not specified, and either NRECS or EREC is specified. The file must be open in exclusive mode.

WPBLKSIZ Indicates the block size of a word processing file. This parameter can be specified as an expression that points to a 1-byte binary number, or a register in parentheses that contains the number. Optional. Ignored if the file is not a word processing file.

WPBLS Indicates the number of bytes in the last sector of a word processing file. This parameter can be specified as an expression that points to a 1-byte binary number, or a register in parentheses that contains the number. Optional. Ignored if the file is not a word processing file.

HXBLK Indicates the block-in-file of the root block of the primary index of the file. This parameter can be specified as an expression that points to a 3-byte binary number, or a register in parentheses that contains the number. Required if PLIST is not specified, and if either DABLK or PTRD is specified. The file must be open in exclusive mode.

DABLK Indicates the block in the file of the starting block of the available block chain. This parameter can be specified as an expression that points to a 3-byte binary number, or a register in parentheses that contains a number. Required if PLIST is not specified, and if either HXBLK or PTRD is specified. The file must be open in exclusive mode.

PTRD Indicates the first data block in file (primary key sequence). This parameter can be specified as an expression that points to a 3-byte binary number, or a register in parentheses that contains a number. Required if PLIST is not specified, and if either HXBLK or DABLK is specified. The file must be open in Exclusive mode.

EXTCOUNT Indicates the count of extents in use by the file. This parameter can be specified as an expression that points to a 1-byte binary number, or a register in parentheses that contains a number. Required if PLIST is not specified, and if either SECEXT or EXTPTR is specified. The file must be open in Exclusive mode. RELEASE=YES is ignored. This parameter is only valid if UPDATFDR issued from privileged code.

SECEXT Indicates the number of blocks in a secondary extent. This parameter can be specified as an expression that points to a 2-byte binary number, or a register in parentheses that contains the number. Required if PLIST is not specified, and either EXTCOUNT or EXTPTR is specified. The file must be open in Exclusive mode. RELEASE=YES is ignored. This parameter is only valid if UPDATFDR issued from privileged code.

EXTPTR Indicates a list of extent pointers. An extent pointer is a pair of three-byte binary numbers: the first contains the block number starting the extent; the second contains the block number plus one ending the extent. The list contains (in order) pairs for the primary, second, and third extents. If pairs for additional extents are appended, the FDR2 is created/modified as necessary. This parameter can be specified as an expression or a register in parentheses that points to the address of the list of pointers. Required if PLIST is not specified, and if either EXTCOUNT or SECEXT is specified. FILE must be open in Exclusive mode. RELEASE=YES is ignored. This parameter is only valid if UPDATFDR is issued from privileged code.

WTFLGS

Indicates the extra write privileges to be granted to the file. This parameter can be specified as an expression or a register in parentheses that points to a 4-byte bit map. Required if PLIST is not specified, and either EXFLGS or RDFLGS is specified. The file must be closed.

RDFLGS

Indicates the extra read privileges to be granted to the file. This parameter can be specified as an expression or a register in parentheses that points to a 4-byte bit map. Required if PLIST is not specified, and either WTFLGS or EXFLGS is specified. The file must be closed.

EXFLGS

Indicates the extra execute privileges to be granted to the file. This parameter can be specified as an expression or a register in parentheses that points to a 4-byte bit map. Required if PLIST is not specified, and either WTFLGS or REFLGS is specified. The file must be closed.

FPCLASS

Indicates the file protection class. This parameter can be specified as an expression or a register in parentheses that points to a 1-byte character field, or as a character string in single quotes, Required if PLIST is not specified, and if CREATOR, CREATE, MODDATE, or EXPDATE specified. The file must be closed.

CREATOR

Indicates the file's owner-of-record. This parameter can be specified as an expression or a register in parentheses that points to a 3-byte User ID, or as a character string in single quotes. Required if PLIST is not specified, and if FPCLASS, CREDATE, MODDATE, or EXPDATE is specified. The file must be closed.

CREDATE

Indicates the creation date of the file. This parameter can be specified as an expression or a register in parentheses that points to a 3-byte packed Julian date field (YYDDD+). Required if PLIST is not specified, and if FPCLASS, CREATOR, MODDATE, or EXPDATE specified. The file must be closed.

MODDATE

Indicates the file's date of last modification. This parameter can be specified as an expression or a register in parentheses that points to a 3-byte packed Julian date field (YYDDD+). Required if PLIST is not specified, and if FPCLASS, CREATOR, CREDATE, or EXPDATE is specified. The file must be closed.

EXPDATE

Indicates the expiration date of the file. An expression or a register in parentheses that points to a 3-byte packed Julian date field (YYDDD+). Required if PLIST is not specified, and if FPCLASS, CREATOR, CREDATE, or MODDATE specified. The file must be closed.

FGPRIORITY Indicates the priority at which the foreground job runs (a value from 1 to 4 with 1 being the highest priority).

BGPRIORITY Indicates the priority at which the background job runs (a value from 1 to 4 with 1 being the highest priority).

Input

One word on top of the stack, as follows:

```
                                              Lower
              |                    |          Address
              |____________________|
   0(SP)  |   |                    |
              | (1) Address of     |          Higher
              |     Parameter List |          Address
              |______Preceding_____|
              |      Stack Data    |
```

(1) The address of the argument list. The format of the argument list is as follows:

	ARGUMENT LIST		
PLIST	(1) Library Name	8 bytes	Lower
	(2) File Name	8 bytes	Address
	(3) Volume Name	6 bytes	
	(4) More Options	1 byte	
	(5) Unused	7 bytes	
	(6) Option Flag	1 byte	
	(7) FDR1NRECS	4 bytes	
	(8) FDR1EBLK	3 bytes	
	(9) FDR1EREC	2 bytes	
	(10) FDR1WPBLKSIZE	1 byte	
	(11) FDR1WPBLS	1 byte	
	(12) FDR1HXBLK	3 bytes	
	(13) FDR1DABLK	3 bytes	
	(14) FDR1PTRD	3 bytes	
	(15) Unused	3 bytes	
	(16) FDR1XTNTCOUNT	1 byte	
	(17) FDR1SECEXT	2 bytes	
	(18) EXTENT LIST PRT	4 bytes	
	(19) FDR1FPCLASS	1 byte	
	(20) FDR1CREATOR	3 bytes	
	(21) FDR1CREDATE	3 bytes	
	(22) FDR1MODDATE	3 bytes	
	(23) FDR1EXPDATE	3 bytes	
	(24) FDR1WTF1GS	4 bytes	
	(25) FDR1RDFLGS	4 bytes	
	(26) FDR1EXFLGS	4 bytes	
	(27) FGPRIORITY	1 bytes	
	(28) BGPRIORITY	1 bytes	Higher
	(29) RESERVED	10 bytes	Address

Group 1 consists of items 7 - 11 above
Group 2 consists of items 12 - 15
Group 3 consists of items 16 - 18
Group 4 consists of items 19 - 23
Group 5 consists of items 24 - 26
Group 6 consists of items 27 - 29

(4) More options:

Bit 0 1 = If the FDR1TXINUSE flag is to be set (ATTACH)
Bit 1 1 = If the FDR1TXINUSE flag is to be turned off (DETACH)
Bit 2 1 = If EXCL LOCK on CLOSE should be set
Bit 3 1 = To update group #6.
Bit 4 Reserved
Bit 5 Reserved
Bit 6 Reserved
Bit 7 Reserved

(5) Unused, must be 0

(6) Option flag:

Bit 0 1 = Set FDR1FLAGSUPDAT (file-closed flag)
Bit 1 1 = Release unused space in extents
Bit 2 1 = Update group 5
Bit 3 1 = Update group 4
Bit 4 1 = Update group 3
Bit 5 1 = Update group 2
Bit 6 1 = Update group 1
Bit 7 1 = Limit file access rights to user's logon access rights for this request

The date of last modification (FDR1MODDATE) is updated in the label whenever a successful UPDATFDR is performed.

(17) The extent list pointer points to a list that contains all the extent information of the file as indicated by FDR1XTNTCOUNT. FDR2 records are created as required to hold this information. The list is constructed as follows:

FDR1X1STRT	3 bytes
FDR1X1END	3 bytes
FDR1X2STRT	3 bytes
FDR1X2END	3 bytes
.	
.	
.	
FDR1XNSTRT	3 bytes
FDR1XNEND	3 bytes

Output

A return code is placed in the top word of the stack replacing input.

```
                                         Lower
              |                    |     Address
              |____________________|
    0(SP) |   |                    |
              |  Return Code       |     Higher
              |____________________|     Address
              |     Preceding      |
              |    Stack Data      |
```

Output

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

Return Codes

Code	Description
0	Success.
4	Insufficient buffer space to perform operation.
8	Volume not mounted.
12	Volume used exclusively by another user or job.
16	Volume has no VTOC.
20	File was not open in Exclusive mode.
24	Library not found.
28	File not found.
32	Insufficient file access rights.
36	FILE was not closed.
40	VTOC full, no space for FDR2.
44	VTOC full, no space for freed extent; extent lost.
48	VTOC error; FDX1 and FDX2 conflict.
52	VTOC error; FDX2 and FDR conflict.
56	VTOC error; FDX1 and FDR conflict.
60	VTOC error; bad data in FDR1 or FDR2.
64	VTOC/system error; FLUB and FDR1 conflict.
68	Disk I/O error; VTOC not reliable.
72	Group 5 update attempted on a file that is not a program file.
76	DMS/TX attach or detach requested and target file is not a DMS/TX file.

Code	Description
80	Remote volume specified.
84	Unused.
88*	Group #3 not valid for volume sets.
92*	Group #7 not valid for single volumes.
96*	Group #8 not valid for single volumes.
100*	Group #9 not valid for single volumes.

* These groups are not available to user programs; they are available to privileged code only.

Examples

```
          UPDATFDR PLIST=(RLIST),VOLUME='SYSTEM',LIBRARY='@SYS001',     -
                FILE='S000',DABLK=DABLK,PTRD=PTRD,HXBLK=HXBLK,          -
                NRECS=NRECS,EREC=EREC,EBLK=EBLK,                        -
                EXTCOUNT=EXTCOUNT,SECEXT=SECEXT,EXTPTR=EXTPTR
+         DS    0H
+         XC    23(7,RLIST),23(RLIST)     . Reserved .
+         MVC   16(6,RLIST),=CL6'SYSTEM'  . MOVE IN VOLUME NAME .
+         MVC   0(8,RLIST),=CL8'@SYS001'  . MOVE IN LIBRARY NAME .
+         MVC   8(8,RLIST),=CL8'S000'     . MOVE IN FILE NAME .
+         MVC   31(4,RLIST),NRECS         . NRECS .
+         MVC   35(3,RLIST),EBLK          . EBLK .
+         MVC   38(2,RLIST),EREC          . EREC .
+         MVC   42(3,RLIST),HXBLK         . HXBLK .
+         MVC   45(3,RLIST),DABLK         . DABLK .
+         MVC   48(3,RLIST),PTRD          . PTRD .
+         MVC   54(1,RLIST),EXTCOUNT      . EXTCOUNT .
+         MVC   55(2,RLIST),SECEXT        . SECEXT .
+         MVC   57(4,RLIST),EXTPTR        . EXTPTR .
+* UPDATFDR RESTRICTED TO USER LOGON ACCESS RIGHTS
+         OI    30(RLIST),15              . OPTION FLAG .
+         PUSH  0,RLIST                   . PARAMETER ADDRESS TO STACK .
+         SVC   25 (UPDATFDR)             . ISSUE UPDATFDR SVC .
```

```
          UPDATFDR PLIST=(RLIST),VOLUME=VSCBNAME,LIBRARY=TESTLIB,       -
                FILE=FDR1FILENAME,FPCLASS='$',CREATOR=FDR1CREATOR,      -
                MODDATE=(8),CREDATE=(9),EXPDATE=FDR1EXPDATE
+         DS    0H
+         XC    23(7,RLIST),23(RLIST)     . Reserved .
+         MVC   16(6,RLIST),VCBSER        . MOVE IN VOLUME NAME .
+         MVC   0(8,RLIST),TESTLIB        . MOVE IN LIBRARY NAME .
+         MVC   8(8,RLIST),FDR1FILENAME   . MOVE IN FILE NAME .
+         MVI   31(RLIST),C'$'            . FPCLASS .
+         MVC   32(3,RLIST),FDR1CREATOR   . CREATOR .
+         MVC   35(3,RLIST),0(9)          . CREDATE .
+         MVC   38(3,RLIST),0(8)          . MODATE .
+         MVC   41(3,RLIST),FDR1EXPDATE   . EXPDATE .
+* UPDATFDR RESTRICTED TO USER LOGON ACCESS RIGHTS
+         OI    30(RLIST),17              . OPTION FLAG .
+         PUSH  0,RLIST                   . PARAMETER ADDRESS TO STACK .
+         SVC   25 (UPDATFDR)             . ISSUE UPDATFDR SVC .
          UPDATFDR PLIST=(R2),VOLUME=VCBSER,LIBRARY=TESTLIB,            -
                FILE=FDR1FILENAME,EXFLGS=FDR1EXFLAGS,RDFLGS=FDR1RDFLAGS,-
                WTFLGS=(4),CLOSE=YES,RESTRICT=NO,RELEASE=YES
+         DS    0H
+         XC    23(7,R2),23(R2)           . Reserved .                03\
+         MVC   16(6,R2),VCBSER           . MOVE IN VOLUME NAME .
+         MVC   0(8,R2),TESTLIB           . MOVE IN LIBRARY NAME .
+         MVC   8(8,R2),FDR1FILENAME      . MOVE IN FILE NAME .
+         MVC   31(4,R2),0(4)             . WTFLGS .
+         MVC   35(4,R2),FDR1RDFLAGS      . RDFLGS .
+         MVC   39(4,R2),FDR1EXFLAGS      . EXFLGS .
+         OI    30(R2),224                . OPTION FLAG .
+         PUSH  0,R2                      . PARAMETER ADDRESS TO STACK .
+         SVC   25 (UPDATFDR)             . ISSUE UPDATFDR SVC .
 FDR1EXFLAGS DC B'10000100'
 FDR1RDFLAGS DC B'11100000'

          UPDATFDR VOLUME='VOLUME',LIBRARY='LIBRARY',FILE='FILE',       -
                HXBLK=(6),DABLK=(7),PTRD=(8),NRECS=(9),EREC=(10),       -
                EBLK=(11)
+         DS    0H
+         PUSHN 0,53                      . SPACE FOR PARAMETER LIST .
+         XC    22(8,15),22(15)           . Reserved .                03\
+         MVC   16(6,15),=CL6'VOLUME'     . MOVE IN VOLUME NAME .
+         MVC   0(8,15),=CL8'LIBRARY'     . MOVE IN LIBRARY NAME .
+         MVC   8(8,15),=CL8'FILE'        . MOVE IN FILE NAME .
+         STCM  9,B'1111',31(15)          . NRECS .
+         STCM  11,B'0111',35(15)         . EBLK .
+         STCM  10,B'0011',38(15)         . EREC .
+         STCM  6,B'0111',42(15)          . HXBLK .
+         STCM  7,B'0111',45(15)          . DABLK .
+         STCM  8,B'0111',48(15)          . PTRD .
```

```
+* UPDATFDR RESTRICTED TO USER LOGON ACCESS RIGHTS
+         MVI   30(15),7                . OPTION FLAG .
+         PUSH  0,15                    . PARAMETER ADDRESS TO STACK .
+         SVC   25 (UPDATFDR)           . ISSUE UPDATFDR SVC .
```

4.2.83 VOL1 - Describe Volume Label

Syntax

```
    VOL1     [NODSECT][,REG=expression][,SUFFIX=character]
```

Function

Describes the standard volume label for disk or magnetic tape. This data structure is the standard volume label for disk or magnetic tapes. The volume table of contents is the primary data structure that leads to the location of files on the storage medium.

Parameter Definitions

NODSECT — Specification of NODSECT results in the VOL1 fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, a DSECT with the name VOL1 (plus optional SUFFIX) is generated.

REG — Provides for the optional specification of a register for which a USING statement for the VOL1 fields in generated.

SUFFIX — If provided, all labels are generated by the concatenation of the letters VOL1, the user-provided SUFFIX (one ASCII character in length), and the field name.

Structure

VOL1

		BYTE 0	BYTE 1	BYTE 2	BYTE 3	
BEGIN	+0	ID				
	+4	SER				
	+8			ACCESS	VSID	
	+C	TOTALEXT				
	+10	FLAGS	RESRV1			
	+14					
	+18					
	+1C					
	+20					
	+24		CREATOR			+25 = OWNER
	+28					
	+2C					
	+30				RESRV2	
	+34					
	+38					
	+3C					
	+40					
	+44					
	+48					
	+4C				LEVEL	
	+50	SYSTEM				
	+54					
	+58	CREDATE			X1STRT	
	+5C			X1END		
	+60		X2STRT			
	+64	X2END			X3STRT	
	+68				X3END	
	+6C		FLG1	FLG2	UCBTYPE	
	+70	VCBBC		VCBMAXTFR		
	+74	VCBCV		VCBCVP		
	+78	MARKER	P1SURF	P1CYL		
	+7C	P1BLOCK				
	+80	P2TOP4				
	+84					
	+88					
	+8C	VCBCVD		XLMTOPEN	XLMTTOTL	
	+90	DSBKNUM				
	+94	DSLENGTH				
	+98	PXSTRT				
	+9C	PXEND				

ORG

+9C	PLOC			
+A0	SDBLK#			
+A4	SDOFFSET			
+A8	SDENTRYLNGTH			
+AC	RESRV5			

Example

```
          VOL1  REG=2
+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 *
+VOL1ID                     DS  CL4'VOL1'   CHARACTERS 'VOL1'
+VOL1SER                    DS  CL6         VOLUME SERIAL NUMBER
+VOL1ACCESS                 DS  C' '        FILE PROTECTION CLASS
+*                                          OR BLANK
+VOL1VSID                   DS  BL1         VOL ID (1-255) IN A SET
+VOL1TOTALEXT               DS  F    TOTAL EXT LIMIT FOR FILE IN MVF
+VOL1FLAGS                  DS  X           ADDITIONAL FLAG
+VOL1MULTIVOL               EQU X'80'       MULTI-VOL FLAG
+VOL1$RESRV1                DS  BL20        RESERVED - ASCII BLANKS
+VOL1CREATOR                DS  CL3         FILE CREATOR ID OR BLANKS
+*                                          FOR MAGNETIC TAPE ONLY
+                           ORG VOL1CREATOR
+VOL1OWNER                  DS  CL14        OWNER ID (OPTIONAL)
+*                                          FOR DISK AND TAPE VOLUMES
+VOL1RESRV2                 DS  BL28        RESERVED - ASCII BLANKS
+VOL1LEVEL                  DS  CL1'1'      MUST BE AN ASCII '1' FOR TAP
+VOL1TAPEEND                EQU *
+VOL1TAPELENGTH           EQU VOL1TAPEEND-VOL1BEGIN
+VOL1SYSTEM                 DS  CL8         SYSTEM IDENTIFICATION
+VOL1CREDATE                DS  PL3         VOLUME INITIALIZATION DATE
+*                                   (PACKED YYDDD+)
+VOL1X1STRT                 DS  FL3         VOLUME TABLE OF CONTENTS 1ST
+*                                  EXTENT STARTING BLOCK ON
+*                                  VOLUME FROM 0
+VOL1X1END                  DS  FL3         FDIR 1ST EXTENT ENDING BLOCK
+*                                  PLUS 1
+VOL1X2STRT                 DS  FL3         VOLUME TABLE OF CONTENTS 2ND
+*                                  EXTENT STARTING BLOCK ON
+*                                  VOLUME FROM 0
+VOL1X2END                  DS  FL3         FDIR 2ND EXTENT ENDING BLOCK
+*                                  PLUS 1
+VOL1X3STRT                 DS  FL3         VOLUME TABLE OF CONTENTS 3RD
+*                                  EXTENT STARTING BLOCK ON
+*                                  VOLUME FROM 0
+VOL1X3END                  DS  FL3         FDIR 3RD EXTENT ENDING BLOCK
+*                                  PLUS 1
+* (EXTENTS 2 AND 3 RESERVED. X2STRT THOUGH X3END MUST CONTAIN
+* BINARY ZEROES.)
+VOL1FLG1                  DS  X          Flag byte 1
+VOL1FLG1CTV               EQU X'80'      Crash tolerant volume
+VOL1FLG1MDTV              EQU X'40'      Media tolerant volume
+VOL1FLG1OLD               EQU X'20'      Old-format volume; all
+VOL1FLGXLMT               EQU X'10'      XTNT limits are set
+*                                        other flags invalid
+VOL1FLG2                  DS  X          Flag byte 2
+VOL1UCBTYPE               DS  AL1        UCB TYPE
+VOL1VCBBC                 DS  AL2        BLOCKS PER CYLINDER
+VOL1VCBMAXTFR             DS  AL2        MAX TRANSFER (BYTES)
+VOL1VCBCV                 DS  AL2        CYLINDERS PER VOLUME
+VOL1VCBCVP                DS  AL2        CYLINDERS PER PHYS VOLUME
+VOL1MARKER                DS  AL1        VS25 pointers follow
+VOL1P1SURF                DS  AL1        Platter # for 1st sector
+*                                        of diagnostic file
+VOL1P1CYL                 DS  AL2        Cylinder # of same
+VOL1P1BLOCK               DS  AL1        And block within track
+VOL1P2TOP4                DS  3A         Pointers for remaining
+*                                        three sectors
+VOL1VCBCVD                DS  AL2        Cylinders/volume incl.
+*                                        diagnostic cylinder
+VOL1XLMTOPEN              DS  XL1        Extent limit for OPEN
+VOL1XLMTTOTL              DS  XL1        Total extent limit
+VOL1DSBKNUM               DS  F          Dump slot block number
+VOL1DSLENGTH              DS  F          Dump slot length
+VOL1PXSTRT                DS  A          Start addr of Page Pool
+VOL1PXEND                 DS  A          End+1 addr of Page Pool
+                          ORG VOL1PXEND
+VOL1PLOC                  DS  X          Rel. loc. of page pool
+                          ORG ,
+VOL1SDBLK#                DS  F          Simple Directory Block #
+VOL1SDOFFSET              DS  H          Offset into block of 1st
+*                                        entry
+VOL1SDENTRYLNGTH          DS  F          Length of single SD entry
+VOL1RESRV5           DS   (256-(*-VOL1BEGIN))C           Filler to end0
+VOL1DISKEND               EQU *
+VOL1LENGTH                EQU 256
+         CSECT
+         USING VOL1,2
```

4.2.84 WPCALL - Call VS Document Access Subroutines

Syntax

```
[label] WPCALL  {OPEN    }    [,DFB={(register)}] [,SUFFIX=character]
                {CLOSE   }          { address  }
                {READ    }
                {REWRITE}
                {DELETE }
                {WRITE   }
                {SEARCH }
                {PRINT   }
                {DOCLIB }
                {STRING }
                {USCORE }
                {XSCORE }
                {UPCASE }
                {LOCASE }

                [,TEXT=string,TEXTLENGTH={(register)}]
                                          {  address }

                [,TEXTOFFSET={(register)}]
                             { address  }
```

Function

Calls VS document access routines to perform I/O operations on a word processing document.

Parameter Definitions

OPEN — Calls the OPEN access subroutine to open a WP file.

CLOSE — Calls the CLOSE access subroutine to close a WP file and updates document summary/header and print information.

READ — Calls the READ access subroutine to read an element or a page of a file.

REWRITE — Calls the REWRITE access subroutine to rewrite an element or a page of a file.

DELETE — Calls the DELETE access subroutine to delete a file.

WRITE — Calls the WRITE access subroutine to write to a file.

SEARCH — Calls the SEARCH access subroutine to perform a character search on a file.

PRINT — Calls the PRINT access subroutine to place a WP file on the queue for printing.

DOCLIB Calls the DOCLIB access subroutine to produce a listing of the document IDs in a document library.

STRING Calls the STRING access subroutine to perform string manipulation on a file.

USCORE Calls the USCORE access subroutine to underscore a string of text.

XSCORE Calls the XSCORE access subroutine to remove underscoring from a string of text.

UPCASE Calls the UPCASE access subroutine to convert a string of text to all uppercase characters.

LOCASE Calls the LOCASE access subroutine to convert a string of text to all lowercase characters.

DFB The address of a data file block (DFB) which is used to pass parameters for the user program to the document access subroutines. If not specified, one DFB with the label DFB is assumed.

SUFFIX One character value that is appended to the DFB to create a unique DFB label.

TEXT Address of a text buffer that contains the text to be manipulated. Used with the READ, REWRITE, WRITE, SEARCH, STRING, USCORE, XSCORE, UPCASE, LOCASE functions.

TEXTLENGTH Contains the number of characters in the text buffer.

TEXTOFFSET Contains the offset from the beginning of the text.

Examples

```
         WPCALL CLOSE

+*       Wang VS Document Access Subroutines - Release 2.00
+*       Program Request for "CLOSE" Function

+        PUSHA 0,DFB              DFB  Pointer in Parameter List
+        OI    0(15),X'80'        Denote Last Parameter
+        LR    1,15               Official Parameter List Pointer
+        JSI   =V(WPCLOSE)        Call Appropriate Subroutine Entry
+        POPN  0,4                Eliminate Parameter List
         WPCALL OPEN

+*       Wang VS Document Access Subroutines - Release 2.00
+*       Program Request for "OPEN" Function
```

```
+         PUSHA 0,DFB                  DFB  Pointer in Parameter List
+         OI    0(15),X'80'            Denote Last Parameter
+         LR    1,15                   Official Parameter List Pointer
+         JSI   =V(WPOPEN)             Call Appropriate Subroutine Entry
+         POPN  0,4                    Eliminate Parameter List


 DOC      WPCALL OPEN,DFB=DOCMNTF,SUFFIX=A

+*        Wang VS Document Access Subroutines - Release 2.00
+*        Program Request for "OPEN" Function

+DOC      PUSHA 0,DOCMNTFA             DFB  Pointer in Parameter List
+         OI    0(15),X'80'            Denote Last Parameter
+         LR    1,15                   Official Parameter List Pointer
+         JSI   =V(WPOPEN)             Call Appropriate Subroutine Entry
+         POPN  0,4                    Eliminate Parameter List
```

4.2.85 WRITE - Write a Record

Syntax

```
[label] WRITE   [{EOM},]UFB={(register)}[,COND=integer]
                 {EOT}      { address  }        15
                 {EOF}
```

Function

Writes one of the following pieces of information:

- The next sequential record to a consecutive or indexed file opened in Output or Extend mode.

- The next sequential record for a consecutive file opened in Shared mode.

- The specified record to an indexed file opened in IO or Shared mode.

For indexed disk files, open in Output 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.

NOTE

The address of the UFB is loaded into register 1.

Parameter Definitions

EOM — Data transmitted by the WRITE function is to be followed with a telecommunications end-of-message character (pertains only to batch telecommunications devices).

EOT — Telecommunications end-of-transmission signal is to be transmitted, following any data specified (pertains only to batch telecommunications devices).

EOF — Write end of file for relative files.

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 the address of the UFB.

COND

If specified, the number or absolute expression becomes the first parameter 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.

Output

The following values are possible result conditions which can be indicated in UFBFS1, UFBFS2 (file status bytes):

- 00 -- Normal completion, success

- 20, 22, and 24 -- Possible invalid-key conditions

  - 21 -- Record sequence error (indexed files only)
  - 22 -- Duplicate key (indexed or relative files only)
  - 24 -- Boundary violation (primary extent size exceeded in output mode - indexed files (or relative files in any mode)

- 30, 34, 95, 96, 97 -- Possible error conditions

  - 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.

Example

```
 OTPUT   WRITE  EOM,UFB=(R1)
+OTPUT   MVI    4(1),B'00000001'     TC WRITE WITH EOM
+        JSCI   15,4(1)              WRITE FUNCTION
         WRITE  EOT,UFB=(R1)
+        MVI    4(1),B'00100000'     WRITE TC EOT SIGNAL
+        JSCI   15,4(1)              WRITE FUNCTION
```

4.2.86 WV46MAP - Describe Parameter List

Syntax

```
    WV46MAP [NODSECT][,SUFFIX=character]
```

Function

Maps the parameter list supplied to the SUBMIT SVC and provides information for use by the SUBMIT macro when using the PLIST option.

Parameter Definitions

NODSECT Specification of NODSECT results in the WV46MAP fields being assembled as part of the current CSECT, DSECT, or STATIC section. If not specified, a DSECT with the name WV46MAP (plus optional SUFFIX) is generated.

REG Provides for the optional specification of a register for which a USING statement for the WV46MAP fields in generated.

SUFFIX If provided, all labels are generated by the concatenation of the letters WV46MAP, the user-provided SUFFIX (one ASCII character in length), and the field name.

Example

```
         WV46MAP
+WV46MAP        DSECT
+*
+*       WV46MAP maps the parameter list supplied to the SUBMIT SVC
+*       (WV46), and provides information for use by the SUBMIT MACRO
+*       when used with the "PLIST" option.  WV46 Uses this map, as
+*       should all callers using the "PLIST" option.
+*
+*       Date  12/03/79       Version 5.00.00
+*
+*********************************************************************
+*
+WV46MAPBEGIN            DS  0F     (Word Alignment Required)
+*
+WV46MAPFILENAME         DS  CL8    Filename of PROC/PRINT File

+WV46MAPLIBRARY          DS  CL8    Library containing FILENAME
+WV46MAPVOLUME           DS  CL6    Volume containing LIBRARY
+WV46MAPCOMDATA          EQU *        (End of Common Section)
+WV46MAPCOMLEN           EQU *-WV46MAPBEGIN
+*
```

```
+* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
+*                   J O B    R E Q U E S T S                          *
+* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
+*
+WV46MAPJOBNAME              DS  CL8    (Optional) Job Name
+WV46MAPJOBCLASS             DS  C      Job Class
+WV46MAPDUMP                 DS  X      Dump Options:  (Action at
+*                                        (Abnormal Termination) :
+WV46MAPDUMPPROG             EQU X'00'      Let Program Decide
+WV46MAPDUMPDUMP             EQU X'C0'      Produce a Dump
+WV46MAPDUMPNONE             EQU X'80'      Do Not Produce a Dump
+*
+WV46MAPLIMIT                DS  F      CPU Execution Time Limit
+*                                       (No Limit if Zero -
+*                                               see FLAGS for Units)
+WV46MAPJSTAT                DS  X      Mode
+WV46MAPJSTATR               EQU X'00'     Run
+WV46MAPJSTATH               EQU X'80'     Hold
+*
+WV46MAPJFLG                 DS  X     Flag Data
+WV46MAPJFLGCHEK             EQU X'80'   Must Be On for LIMIT
+WV46MAPJFLGCNCL             EQU X'40'   Cancel (at Expiration)
+WV46MAPJFLGPAUS             EQU X'20'   Pause  (at Expiration)
+WV46MAPJFLGREQ              EQU X'04'   REQUEUE after execution
+WV46MAPJFLGSEC              EQU X'01'   LIMIT Units are in Seconds
+*                                         (Else units are in Clock Units
+*                                          ** NOTE ** Starting with
+*                 ** Note **              Release 6.0 Clock Units will
+*                                         no longer be acceptable)
+WV46MAPJTYPE                DS  X      Job Type                     02\
+WV46MAPPERMANENT            EQU X'80'     Permanent job             02\
+WV46MAPJSPARE               DS  XL5   * Reserved * Must Be Zero     01\
+*
+WV46MAPJLENGTH              EQU *-WV46MAPBEGIN        Map Length (Job)
+* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
+*                  P R I N T    R E Q U E S T S                       *
+* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
+*
+        ORG   WV46MAPCOMDATA
+*
+WV46MAPPRTCLASS           DS  C       Print Class
+WV46MAPFORM#              DS  BL1     Form #
+WV46MAPCOPIES             DS  BL2     # Copies
+*
+WV46MAPPSTAT              DS  X       Status Indicator
+WV46MAPPSTATH             EQU X'80'     Hold
+WV46MAPPSTATS             EQU X'00'     Spool
+*
+WV46MAPDISP               DS  X       Disposition
+WV46MAPDISPSCR            EQU X'00'     Scratch
```

```
+WV46MAPDISPREQ            EQU X'40'     Requeue
+WV46MAPDISPSAV            EQU X'20'     Save
+*
+WV46MAPPSPARE             DS  XL16    * Reserved *  Must Be Zero
+*
+WV46MAPPLENGTH            EQU *-WV46MAPBEGIN        Map Length (Print)
+* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
+*  T R A N S M I T / R E T R I E V E   R E Q U E S T S            *
+* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
+         ORG   WV46MAPBEGIN
+*
+WV46MAPOISNAME              DS  0CL71  OIS File Namestring
+*
+WV46MAPDOCID                DS  0CL5   Document ID Name
+WV46MAPDOC#                 DS  CL4    Document Number
+WV46MAPDOCLIB               DS  CL1    Document Library
+WV46MAPPASSWRD              DS  CL6    Document Password
+*
+WV46MAPRESVD1               DS  CL5    Reserved - must be zero
+*
+WV46MAPDOCVOL               DS  CL6    Document Volume
+*
+         ORG   WV46MAPCOMDATA
+*
+WV46MAPRESVF1               DS  CL49   Reserved - must be zero
+*
+WV46MAPROISNAME             DS  0CL71  Remote OIS File Namestring
+*
+WV46MAPRDOCID               DS  0CL5   Remote Document ID Name
+WV46MAPRDOC#                DS  CL4    Remote Document Number
+WV46MAPRDOCLIB              DS  CL1    Remote Document Library
+WV46MAPRPASSWRD             DS  CL6    Remote Document Password
+WV46MAPRDOCVOL              DS  CL8    Remote Document Volume
+*
+WV46MAPRESVR1               DS  CL52   Reserved - must be zero
99
+         ORG   WV46MAPROISNAME
+WV46MAPRFILENAME            DS  CL8    Remote Filename
+WV46MAPRLIBRARY             DS  CL8    Remote Library
+WV46MAPRVOLUME              DS  CL6    Remote Volume
+WV46MAPRESVR2               DS  CL49   Reserved - must be zero
+*
+WV46MAPNAMTYPE              DS  X      Local Filename Format
+WV46MAPNAMUNDEF             EQU X'00'     Undefined
+WV46MAPNAMVS                EQU X'01'     VS Format
+WV46MAPNAMDOC               EQU X'02'     Document Format
+WV46MAPNAMVSOIS             EQU X'03'     VS (OIS) Format
+WV46MAPNAMOIS               EQU X'04'     OIS Format
+WV46MAPRNAMTYPE             DS  X      Remote Filename Format
+WV46MAPRNAMUNDEF            EQU X'00'     Undefined
```

```
+WV46MAPRNAMVS               EQU X'01'     VS Format
+WV46MAPRNAMDOC              EQU X'02'     Document Format
+WV46MAPRNAMVSOIS            EQU X'03'     VS (OIS) Format
+WV46MAPRNAMOIS              EQU X'04'     OIS Format
+*
+WV46MAPLOCATION             DS  CL8    Local or Remote Location Name
+                            DS  CL8    Reserved - must be zero
+WV46MAPGROUP                DS  CL16   Transfer Group
+                            DS  CL16   Reserved - must be zero
+*
+WV46MAPREPLACE              DS  X      Replace
+WV46MAPREPLACEN             EQU X'00'     Abort on Duplicate File
+WV46MAPREPLACEY             EQU X'80'     Scratch on Duplicate File
+*
+WV46MAPSTATUS               DS  X      Queue Status
+WV46MAPSTATUSA              EQU X'00'     Active
+WV46MAPSTATUSH              EQU X'80'     Hold
+*
+WV46MAPDISPOS               DS  X      Disposition
+WV46MAPDISPSAVE             EQU X'00'     Save
+WV46MAPDISPSCRA             EQU X'80'     Scratch
+*
+WV46MAPXFERDISP             DS  X     Transfer Disposition
+WV46MAPXFERDST              EQU X'00'     Store
+WV46MAPXFERDPR              EQU X'01'     Print
+WV46MAPXFERDRN              EQU X'02'     Run
+*
+WV46MAPCOMDATA1             EQU *    (End of common section)
+*
+         ORG   WV46MAPCOMDATA1
+*
+*        FILE TRANSFER PRINT OPTIONS
+*
+WV46MAPRPRTCLAS             DS  C      Print Class
+WV46MAPRFORM#               DS  BL1    Form #
+WV46MAPRCOPIES              DS  BL2    # Copies
+WV46MAPRPRTDISP             DS  X      Print Disposition
+WV46MAPRPRTSCR              EQU X'00'     Scratch
+WV46MAPRPRTREQ              EQU X'40'     Requeue
+WV46MAPRPRTSAV              EQU X'20'     Save
+WV46MAPPRTMODE              DS  X      Print Mode Status
+WV46MAPPRTMODES             EQU X'00'     Spool
+WV46MAPPRTMODEH             EQU X'80'     Hold
+*
+*        ADDITIONAL  WP  DOCUMENT PRINT OPTIONS
+*
+WV46MAPSTART                DS  X      Print from Page
+WV46MAPFINISH               DS  X      Print thru Page
+WV46MAPNUMBER               DS  X      Start as Page Number
+WV46MAPHEADER               DS  X      First Header Page
```

```
+WV46MAPFOOTER             DS   X      First Footer Page
+WV46MAPLINE               DS   X      Footer begins on Line
+WV46MAPPGLTH              DS   X      Page Length
+WV46MAPCSET1              DS   X      Character Set 1

+WV46MAPCSET2              DS   X      Character Set 2
+WV46MAPPRINTER            DS   X      Printer Number
+WV46MAPMARGIN1            DS   X      Left Margin 1
+WV46MAPMARGIN2            DS   X      Left Margin 2
+WV46MAPPITCH              DS   X      Pitch
+WV46MAPPITCH10            EQU  X'01'     10
+WV46MAPPITCH12            EQU  X'02'     12
+WV46MAPPITCHPS            EQU  X'03'     PS
+WV46MAPPITCH15            EQU  X'04'     15
+WV46MAPFMAT               DS   X      Format
+WV46MAPFMATUNJ            EQU  X'00'     Unjustified
+WV46MAPFMATJUS            EQU  X'80'     Justified
+WV46MAPFMATNOT            EQU  X'40'     With Notes
+WV46MAPFRMS               DS   BL1    Forms
+WV46MAPFRMSCON            EQU  X'00'     Continuous
+WV46MAPFRMSSTD            EQU  X'80'     Standard
+WV46MAPFRMSFM1            EQU  X'40'     Form 1
+WV46MAPFRMSFM2            EQU  X'20'     Form 2
+WV46MAPSTYLE              DS   X      Style
+WV46MAPSTYLEFIN           EQU  X'00'     Final
+WV46MAPSTYLEDRF           EQU  X'80'     Draft
+WV46MAPSUM                DS   X      Summary
+WV46MAPSUMOMIT            EQU  X'00'     Omit
+WV46MAPSUMPRT             EQU  X'80'     Print
+WV46MAPLINES              DS   X      Lines
+WV46MAPLINES6             EQU  X'00'     6 Per Inch
+WV46MAPLINES8             EQU  X'80'     8 Per Inch
+WV46MAPRESVS2             DS   XL4    Reserved - must be zero
+*
+          ORG   WV46MAPCOMDATA1
+*         FILE TRANSFER RUN OPTIONS
+*
+WV46MAPJNAME              DS   CL8    Job Name
+WV46MAPJMODE              DS   X      Job Mode
+WV46MAPJMODER             EQU  X'00'     Run
+WV46MAPJMODEH             EQU  X'80'     Hold
+WV46MAPJDISP              DS   X      Job Disposition
+WV46MAPJDISPRQ            EQU  X'80'     Requeue
+WV46MAPACT                DS   X      Job Action
+WV46MAPACTWARN            EQU  X'00'     Warn
+WV46MAPACTCNCL            EQU  X'80'     Cancel
+WV46MAPACTPAUS            EQU  X'40'     Pause
+WV46MAPJCLASS             DS   C      Job Class
+WV46MAPJLIMIT             DS   F      CPU Execution Time Limit
+*                                      (No Limit if Zero)
```

```
+WV46MAPJDUMP             DS  X      Dump Options:  (Action at
+*                                    (Abnormal Termination) :
+WV46MAPJDMPPROG          EQU X'00'       Let Program Decide
+WV46MAPJDMPDUMP          EQU X'C0'       Produce a Dump
+WV46MAPJDMPNONE          EQU X'80'       Do Not Produce a Dump
+WV46MAPRESVD2            DS  XL11   Reserved - must be zero
+         ORG
+WV46MAPLENGTH            EQU *-WV46MAPBEGIN         Map Length (T/R)
```

4.2.87 XIO - Execute Physical I/O (SVC 3)

Syntax

Format 1:

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

Format 2:

```
[label] XIO OFB={ address  },COMMAND={ address  },
                {(register)}         {(register)}

            MEMA={ address  },BLKNUM={ address  },
                 {(register)}        {(register)}

            BLKSIZE={ address  }[,RELEASE]
                    {(register)}

            [,VOLIO={YES},VCB={ address  }]
                    { NO}     {(register)}

            [,MLPRINT={YES},FORM={LIST}]
                      {NO }      {EXEC}

            [,UCPRINT={YES}][,DEVSTATUS={CLEAR  }]
                      {NO }             {CHECK  }
                                        {NOCHECK}

            [,DIAG={YES}][,PRIORITY={YES}][,PAGEMARK={YES}]
                   {NO }            {NO }            {NO }
```

Function

XIO performs the following functions:

- 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(s) in physical pages during the I/O operation.
- Constructs the IOCW.
- 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 the read-type I/O is accomplished.

- Enters the system start I/O routine to initiate the operation.

- 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 the IOCW (from COMMAND parameter, converted MEMA parameter, converted BLKNUM parameter) 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.

NOTE

For both normal disk file I/O and for VOLIO, under the nonstandard addressing (NSA) option for soft-sectored diskettes, the user program calculates sector sizes and addresses, and passes to the XIO SVC the sector addresses for each I/O operation. Under the NSA option, the XIO SVC skips address (extent) validation and the usual block-to-pseudo-sector translation.

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.

The following restrictions on general I/O capability are enforced by the XIO routine:

- All

  - All memory addresses for a READ or WRITE operation must be valid (present in main memory or page faulted) and must be in the user-modifiable data area (unless the requesting routine is privileged) as an I/O buffer area or entirely above the XIO parameter list on the stack.

- Disk

  - A block to be read or written must fall within the current extent limits of the specified file (except for VOLIO disk requests).

  - The specified memory address must be on a page boundary.

  - The VOLIO option (FLAG bit 1 = 1) is allowed only when requested from within System Mutual Exclusion (SME) state.

  - The length specified for a READ or WRITE operation must be a multiple of the page size.

  - Indirect data addressing is always used for disk I/O.

- Library-structured diskette

  - All restrictions as for other disk.

- Unstructured diskette

  - 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.

  - The VOLIO option is ignored.

NOTE

A nonstandard 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 calculate the sector size and addresses, set mode, and set density. When nonstandard addressing is specified, the XIO SVC does not perform extent validation or address translation, but simply passes the address to the firmware via the I/O control word (IOCW).

- Tape

  - The maximum size permitted for tape records is 32K.

- Printer

  - 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 cannot be less than 2 or more than 134. The data for a block print operation must be on a single page.

  - 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

  - An attention identification (AID) character is stored in the current status portion of the device's Unit Control Block (UCB) on successful completion of each I/O operation. (See the 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.

  - If the device's UCB indicates that the keyboard is unlocked when a READ operation is requested, the XIO routine waits for an attention interruption from this device. When such an interruption is received, the interrupt service routine marks the UCB keyboard locked and then allows XIO to initiate the read operation.

  - Indirect data addressing is always used by XIO for workstation I/O.

Parameter Definitions

PLIST The address of a 16-byte area that contains the parameter list for XIO. If this parameter is supplied, any other parameters are used to modify the parameter list after it has been moved to the stack. The original copy is not modified.

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. The VOLIO parameter is not used with this parameter.

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 A virtual data address to be translated to a physical address and then placed in the IOCW for the I/O operation. This parameter can be specified as an address expression that points to a 4-byte area which contains the virtual address in its low-order three bytes, or as a register specification in parentheses where the register contains the virtual address.

BLKNUM For disk I/O, the address of a 3-byte area that contains the number of the block to be read from the file. If the VOLIO option is specified, or if an unstructured diskette device is being referenced, this is to be the block on volume, from block 0.

BLKSIZE The address of a halfword that contains the length in bytes of the data to be transferred (or maximum length, as for magnetic tape).

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 volumes, 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 is modified if the value of the parameter is an address.

MLPRINT YES requests a block I/O operation to the printer 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 stacked. The supervisor call is generated. If other parameters are supplied, they are used to modify the existing parameter list. The VOLIO=YES and RELEASE parameters 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 parameter is normally not useful on an XIO macroinstruction with FORM=LIST.

UCPRINT If YES is specified, then uppercase printing is used. The default is NO.

DEVSTATUS This parameter is intended for the use of hardware diagnostics personnel when simulating error conditions on serial workstations and printers.

If CLEAR is specified, XIO resets 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, any attempts at I/O to a malfunctioning workstation cause the task to wait for the device to become operational. NOCHECK is the default value.

DIAG When DIAG=YES, diagnostic mode and READ/WRITE ECC are enabled. Then, the control commands SEEK and FORMAT will be permitted in the command byte of the IOCW. Before issuing the XIO, the user must issue a call to GETHEAP to allocate a 2-page buffer. The XIO MEMA parameter must also be specified with the address of the buffer. Caller must be privileged and have diagnostic authorization specified through the Security program.

PRIORITY Specyifying YES marks this I/O as a priority I/O. This allows the reading of the label of a newly spun-up disk without letting regular I/O through. This is a privileged function and is only valid for a VOLIO operation.

PAGEMARK Specifying YES causes the related pages in memory to be marked as having no valid information only if the I/O operations was successful. The pages will not be swapped out to the paging file.

Stack On Input

The top 16 bytes of the stack are a parameter list that contains:

```
                                             Lower
            |                     |          Address
            |0    1     2     3   |
                 ___________________
  0(SP) |   |     |               |
            | (1) | (2) OFB       |
            |_____|______Address__|
  4(SP)     |     |               |
            | (3) | (4) IOCW      |
            |_____|______Address__|
  8(SP)     |              |     |    |
            | (5) Length   |(6)  | (7)|
            |______________|_____|____|
 12(SP)     |                    |    |
            | (8) Block Number   | (9)|      Higher
            |____________________|____|      Address
            |      Preceding          |
            |     Stack Data
```

(1) Flag byte:
Bits 0-7 = X'FF', then there are two flag bytes (Notes 6 and 7 below). If bits 0-7 do not equal X'FF', the flag byte is as follows:

Bit 0 Reserved, must be zero

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 to 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.

(2) OFB address: address of open file block (OFB) for file. When bit 7 of the flag bit (described in Note 1) = 1 for telecommunications option, this value contains the address for the VS-DLP communication path. When bit 1 of the flag bit (described in Note 1) = 1 for VOLIO option, this value is the address of the volume control block (VCB) for the disk volume.

(3) Command byte for I/O command word (IOCW).

(4) Memory address (virtual) for IOCW if a read or write command is in byte 4.

(5) Length in bytes for operation (read or write command, all devices).

(6) Flag byte 1 -- if (1) above contains X'FF', then this byte is the first flag byte and is formatted as shown above in (1). Otherwise this byte is unused.

(7) Flag byte 2 -- if (1) above contains X'FF', then this byte is the second flag byte (otherwise the byte is unused) and is formatted as follows:

Bit 0	1 = Diagnostic option
Bit 1	1 = Mark the pages related to the I/O operation as NO VALID INFORMATION if and only if the I/O operation was successful.

(8) 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). If the VOLIO option is selected, this value contains the block number (in binary) within the volume of first block to be read or written, (where first block of a volume is block 0). If the telecommunications option is selected, then this value contains bytes 6-8 of IOCW (for disk files only).

(9) Unused.

Stack On Output

Return codes are placed in the top word of the stack (replacing the input parameters.

- Low-order halfword of return code field - binary return codes
- High-order halfword of return code field - residual block counts
  - Return codes 4, 8 -- specified block size minus number of bytes actually read or written.
  - Other return codes -- always 0.

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 other return codes are set, the operation has been suppressed.

```
                                                Lower
             |                      |           Address
             |______________________|
    0(SP) |  |                      |
             |     Return Code      |
             |______________________|
    4(SP)    |                      |
             |                      |           Higher
             |______________________|           Address
             |       Preceding      |
                    Stack Data
```

Return Codes

Code Description

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 be either in the 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 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.

Code	Description
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.
48	Diagnostic pages are not fixed in physical memeory.
52	Unable to complete remote XIO request.

Example

```
 LAB1     XIO    COMMAND=RDCMD,PLIST=XIOPARM
+LAB1     PUSHC  0(16,0),XIOPARM
+         MVC    4(1,15),RDCMD
+         SVC    3 (XIO)

 LAB2     XIO    OFB=(R1),MEMA=(R2),BLKNUM=UFBBUFBLOCK,                   X
                 BLKSIZE=UFBBLKSIZE,COMMAND=WRCMD,RELEASE
+LAB2     PUSHN  0,16
+         XC     0(16,15),0(15)
+         MVC    8(2,15),UFBBLKSIZE
+         MVC    12(3,15),UFBBUFBLOCK
+         STCM   R2,7,5(15)
+         MVC    4(1,15),WRCMD
+         STCM   R1,7,1(15)
+         MVI    0(15),B'10000000'
+         SVC    3 (XIO)
          XIO    COMMAND=RDCMD,PLIST=XIOPARM,VOLIO=YES
+         PUSHC  0(16,0),XIOPARM
+         MVC    4(1,15),RDCMD
+         OI     0(15),B'01000000'
+         SVC    3 (XIO)
```

4.2.88 XMIT - Transmit Intertask Message (SVC 36)

Syntax

```
[label] XMIT    MESSAGE={(register)},PORT={(register)}
                        { address  }      { address  }
                        { 'string' }

                [,NOWAIT][,OTHERTASK]
```

Function

Transmits a message between user tasks, or between a user task and a specific subsystem of the operating system. The message supplied on input to the SVC is placed in a system message buffer. The message is then copied from the system message buffer to the address specified by the receiver as a result of the CHECK SVC routine with the MESSAGE option. The CHECK macroinstruction is used to accept receipt of a message.

Parameter Definitions

MESSAGE The address of a message, which may be stored 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 2048. This parameter can be specified as a register in parentheses that contains the address of the message, or as an expression that addresses the message.

PORT The 4-character name of the receiving message port. The value of this parameter can be an address expression, a register designation where the register contains the address of the four characters in memory, or a character string in quotes.

NOWAIT If specified, control returns to issuer immediately if there is insufficient space in the receiving port's message buffer to insert the message.

OTHERTASK If specified, control returns to issuer immediately if the designated receiving message port belongs to the transmitting task.

Stack On Input

```
                                                    Lower
                |                      |            Address
                |0     1     2     3   |
      0(SP)  |  |______|_______________|
                | (1)  |(2) Message    |
                |      |    Address    |
      4(SP)     |______________________|
                | (3) Name of Message  |            Higher
                |     Receipt Port     |            Address
                |______________________|
                |      Preceding       |
                |     Stack Data       |
```

(1) Flag byte:
Bit 0 0 = WAIT until there is enough buffer space if there is not enough at the time.
1 = NOWAIT option, return to caller if there is not enough buffer space.
Bits 2-7 0 = OTHERTASK option, transmit only to other tasks.

(2) Address of a message to be transmitted -- the first two bytes of the supplied message indicate its length, including those bytes, and can not be greater than 2048.

(3) Name of the message receipt port -- 4-character string.

Stack On Output

```
                                                    Lower
                |                      |            Address
                |______________________|
      0(SP)  |  |                      |
                |     Return code      |
                |______________________|
      4(SP)     |                      |
                |     Unused           |            Higher
                |______________________|            Address
                |      Preceding       |
                |     Stack Data       |
```

Output

Return codes are placed in the top word of the stack.

Return Codes

Code	Description
0	Success.
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 transmitting task.
20	Port is too small to accept this message.

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
THE USER PROGRAM

5.1 INTRODUCTION

This chapter discusses the user program and the concepts that relate to the development of programs within the VS operating system environment. The term "user program" refers to all programs that are developed with the VS operating system. This includes application programs such as general ledger systems, order entry/inventory systems, payroll/personnel systems, as well as Wang-supplied system programs.

All programs must follow the program development procedure before they can be run on the system. Included in this procedure is the translation of the source language statements into machine language, resolution of addresses, linking of independently written modules, and generation of runtime information. Once these steps are accomplished, the program is ready to run. However, completion of these steps does not guarantee that the program will produce the desired results. A testing and debugging phase is always necessary.

A program is a sequence of coded statements properly prepared to run on the operating system, that the command processor or the LINK SVC can invoke. The following sections describe the structure of the user program, its development, and its running environment.

5.2 THE PROGRAM DEVELOPMENT PROCESS

No matter what language a program is written in, there are steps that must be accomplished before the program can run on the VS operating system. The steps include

- Problem definition and coding.
- Translation of the code into an object module that may include linking independent modules. Several code modules may be linked together to produce a program.
- Running, testing, and debugging the code.

5.2.1 Problem Definition and Coding

Every computer program is created to solve a problem. For example, a payroll package solves the problem of how to quickly and efficiently issue payroll checks and maintain employee salary records.

In this phase of the program development process, the software developer must clearly define the problem, design a solution that solves the problem, and then code the solution in the language best suited to the application. An Editor is used to create a source file that contains the code for the program.

5.2.2 Translation of the Code

The source file is then processed through a language translator (for example, an assembler or a compiler) which checks for syntax errors, translates the source code into machine language, and creates a program listing file. This file, which contains the program object module that is formatted for execution, also contains machine instructions, data storage areas, and other information that pertains to the program.

A program can be constructed of individually assembled or compiled source files, called modules, or may be complete in one source module. If the program is comprised of more than one module, the modules must be linked together using the VS LINKER utility. The Linker defines pointer addresses and provides information for proper transfer of control from one module to another.

5.2.3 Running, Testing, and Debugging the Program

Once the program is coded and the object module created, the programmer must run the program to be sure that the program is logically correct. If the program fails to perform the function for which it was designed, it must be debugged. When the problem is found and a solution decided upon, the process begins again. The programmer enters the changes to the source code using the Editor, reassembles or recompiles it, links it, and then reruns and retests the program.

5.3 STRUCTURE OF THE PROGRAM FILE

The language translator generates an object program with two sections: a reentrant section and a modifiable section.

5.3.1 The Reentrant Section

The reentrant section contains the machine instructions that comprise the logic of the program. It contains no modifiable code or data. This means that many users can share one copy of the code loaded into memory without affecting the results of another user. When a program is run, the reentrant section of the program object file is mapped into a task's program code area address space.

5.3.2 The Modifiable Section

Although many users may share the same code, each user has a separate modifiable area used to store variable data and dynamically initialized variables. This area is called the modifiable data area. It contains the program stack, and an I/O buffer area (or heap). Section 5.4 describes the user's modifiable data area.

5.4 THE USER'S MODIFIABLE DATA AREA

When a program is run, the reentrant section of the program object file is mapped into a task's program code area address space. The modifiable section of the object program maps into the task's modifiable data area and is used for the program stack and buffer areas.

The user's modifiable data area is divided into two sections, a stack area and a buffer area. The stack area starts at the highest address and grows downward. The buffer area (or heap area) starts at the lowest address and grows upwards. Figure 5-1 shows the layout of the modifiable data area.

The VS instruction set includes several stack-oriented instructions that can affect the modifiable data area stack. A push operation decrements the stack pointer (register 15, known as SP) as long as the resulting stack pointer does not cross the control register 2 value. Control register 2 (CR2) indicates the end of the buffer area; therefore, the stack pointer cannot go beyond it. Attempts to push the stack pointer beyond (lower than) the stack limit results in a stack overflow program check, which usually cancels the program.

The area between the stack limit and the stack pointer should never be referenced, since it lies neither in the stack area nor in the buffer area.

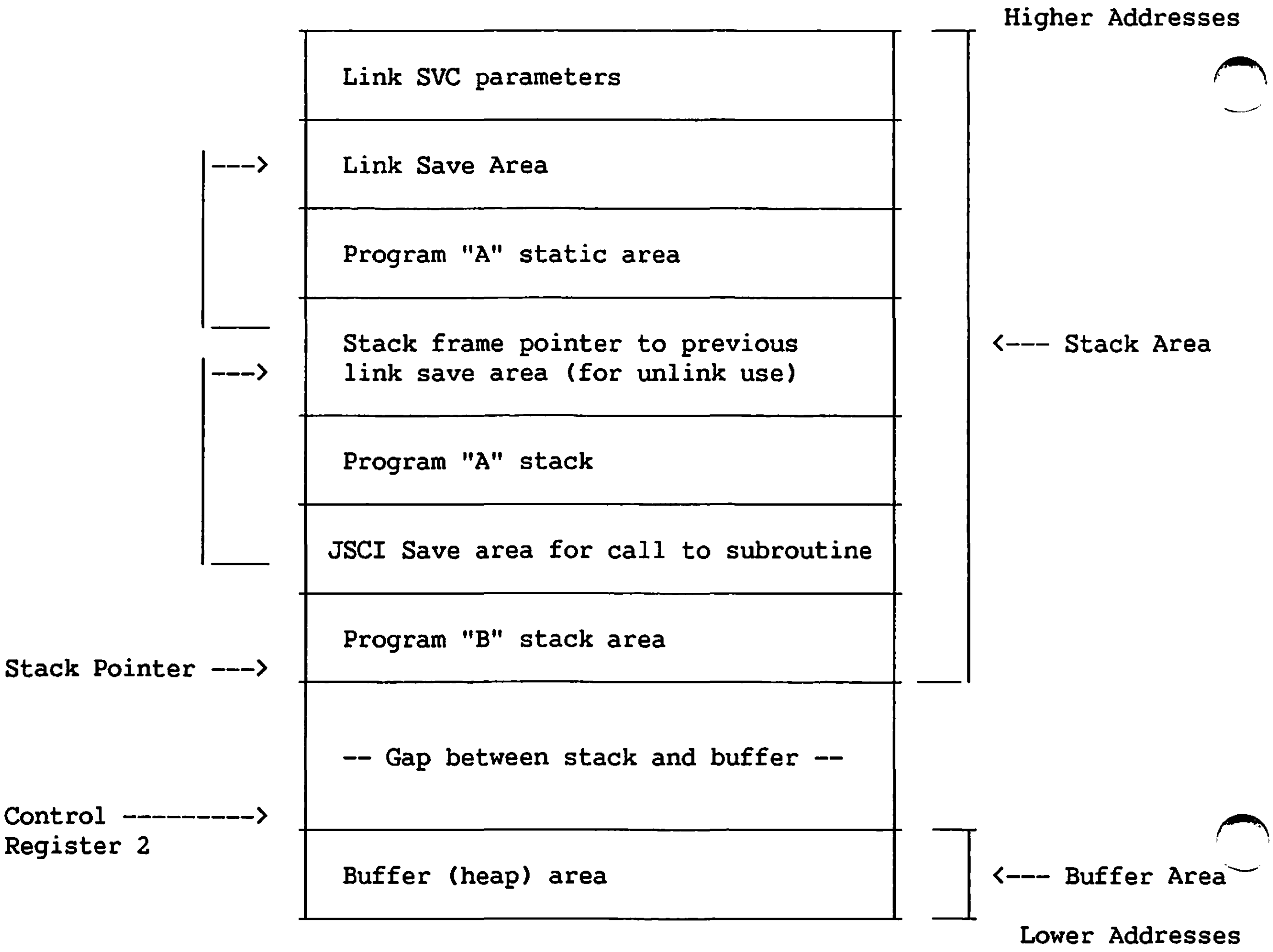


Figure 5-1. The User's Modifiable Data Area

5.4.1 JSCI, SVC, and LINK Save Areas

In addition to program data, the stack is used to save linkage information such as registers or the PCW (program control word) at each program, procedure, or subroutine invocation.

The JSCI (Jump to Subroutine on Condition Indirect) instruction is used for subroutine calls. The information pushed onto the stack by the JSCI instruction is shown in Figure 5-2. Control register 1 is used to maintain a chain of save areas, also known as stack frames. The subroutine can modify the stack pointer (SP) by pushing additional information onto the stack. The RTC (return) instruction resets the stack pointer to point to the area following the stack frame (the save area) by setting it to the value in control register 1. It then pops the registers and control register 1 off the stack and branches to the return address, thus returning from the subroutine.

```
                                                      Lower Address
 ______________________________________________
| Register 0 Save                              |
| Register 1 Save                              |
 ----------------------------------------------
                    .
                    .
                    .
 ______________________________________________
| Register 14 Save                             |
| 0000PPP0   Control Reg 1 Save (Call Chain)   |
| MASK  Return Address                         |   Higher Address
 ----------------------------------------------
```

Figure 5-2. JSCI Save Area

The SVC instruction manages the stack somewhat differently. The entire program control word (PCW) is pushed onto the stack, rather than just the return address (See Figure 5-3). The SVCX instruction pops all the information from the stack and returns control to the SVC caller. Most SVC calling sequences push parameter information onto the stack before the SVC instructions are issued. The SVCX instruction pops these parameters off the stack by letting the SVC routine specify the new stack pointer before issuing the SVCX instruction.

```
                                                      Lower Address
 ______________________________________________
| Register 0 Save                              |
| Register 1 Save                              |
 ----------------------------------------------
                    .
                    .
                    .
 ______________________________________________
| Register 14 Save                             |
| 0000PPP1   Control Reg 1 Save (Call Chain)   |
| SVC #  Old PCW Address                       |
| Old PCW Status Bits                          |
| SVC Parameters                               |   Higher Address
 ----------------------------------------------
```

Figure 5-3. SVC Save Area

One program can call another by invoking the LINK SVC. In this case, the called program's static area is pushed onto the stack and initialized with information from the object file. This area is called static because it remains for the life of the program, whereas the program can allocate temporary (dynamic) storage by pushing data onto the stack and popping it when done. When control is given to the program, register 14 points to the static area.

After pushing and initializing the static area, the LINK SVC constructs a JSCI-type save area, where the return address points to an instruction that calls the UNLINK SVC. This allows programs and subroutines to issue an RTC instruction (or RETURN macro) to return to their caller. Finally, LINK constructs a SVC-type save area so that it can return to the new program via SVCX.

At UNLINK time, program resources are deallocated, control register 1 is set to point to the LINK save area, and SVCX is used to return to the previous program.

The VS DEBUGGER utility provides a Trace command which allows the user to examine the chain of CALL (JSCI), LINK, and SVC save areas.

5.4.2 Buffer Management

The low end of the modifiable data area is used as a buffer and heap area. Buffer and heap management is provided by the GETBUF, FREEBUF, GETHEAP, and FREEHEAP SVCs.

GETBUF and FREEBUF allocate page-aligned buffers which are multiples of 2 KB (kilobytes) long.

GETHEAP and FREEHEAP are the recommended SVCs for buffer and heap management. They provide a flexible and efficient memory management system because blocks or heaps of any size may be allocated. Also, blocks are organized into subpools, to allow blocks in the same subpool, obtained via separate calls to GETHEAP, to be freed with one call to FREEHEAP. Finally, subpools are associated with a link level (usually the current one) and are automatically freed when unlinking from that link level.

5.5 TRANSFER OF PROGRAM CONTROL

An assembly program invokes subroutines and programs in the following ways:

- The BAL, BALR, BALCI, BALS, JSCI, or JSI instructions are used within a program to save a return point in a register or on the stack and then enter a subroutine. The BC, BCR, BCS, or RTC instructions are used to return.

- 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, the supervisor executes an SVCX instruction to return to the user program. The user program does nothing more than place the address of required arguments, or in some instances the arguments themselves, on the top of the stack and issue the SVC instruction. Routines entered by the SVC instruction normally remove their input arguments and leave output information on the top of the stack.

- The JSCI or JSI instruction is used to start a transfer of control between routines that are linked into a single program either before runtime (statically) or during runtime (dynamically). The CALL macroinstruction generates the JSCI instruction. Refer to Chapter 4 for an example of the expanded CALL macroinstruction showing the sequence of instructions that perform this transfer of control. Figure 5-2 shows the save area placed on the stack by the JSCI instruction.

- The LINK SVC is used to transfer control between programs that were not bound together by the VS LINKER utility. The linked-to program should return to the issuing program by means of the RETURN macroinstruction. When the LINK SVC is invoked, the linked-to program's static area is placed on the stack area of the linking program (Refer to Figure 5-1). Figure 5-3 shows additional information that is placed on the stack in the SVC save area.

5.6 INTERACTING WITH THE WORKSTATION

The workstation is the primary data entry and display device used with the VS operating system. It is also used as a system console device for the presentation of and response to solicited or unsolicited system messages. Communication between the user and the workstation is managed through the GETPARM SVC.

In addition to sending messages to the workstation, the GETPARM facility accesses runtime parameters such as device or file assignments, batch-oriented runtime option lists, or interactive program data. These runtime parameters are either obtained directly from the user at the workstation or come from a procedure file. Each parameter requested from the user must be labeled with a parameter reference name (prname). Programs that require a more flexible level of interaction than that provided by GETPARM should access the user workstation using the standard data management facilities. However, to be usable with the procedure language, programs must access all runtime parameters using the GETPARM SVC. Also, the PUTPARM macroinstruction allows a program to pass GETPARM parameter values to another program that is being invoked through the LINK SVC.

Whenever the user enters GETPARM processing, the workstation screen, its resident buffer contents, its status (keyboard locked or unlocked, attentions received, etc.), and its tab settings are saved in the task's current stack. Before resuming user program processing, this screen and status are restored. This process also occurs when the user enters the Help Processor by pressing the HELP key on the workstation.

Messages sent to the workstation must be in the format specified in the CANCEL and GETPARM SVC descriptions described in Chapter 4. Messages that relate to background program runs (programs running without an associated workstation) are sent to the operator console.

Workstation processing can also be achieved by treating the workstation as a file. By using the VS Data Managment System (DMS), users can write applications that can read and write to the workstation. DMS workstation screen interaction is supported in BASIC, COBOL, PL/I, RPGII, and Assembler. You must establish a User File Block (UFB) in your program for each file accessed by DMS. Refer to the VS Data Management System Reference for more information on DMS and on creating a UFB.

5.7 STANDARD PRNAMES

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 functions. Within groups of related programs, naming conventions should be established to enhance recognition and predictability.

The following list contains standard prnames used by system utility programs and compilers:

INPUT	The input file
INPUT1-INPUT(n)	Multiple input files used for related purposes
OUTPUT	The output file
LIBRARY	A library used for input purposes
WORK	An I/O file used for temporary storage
WORK1-WORK(n)	Multiple I/O files used for temporary storage
OPTIONS	The batch-oriented option list used to define runtime parameters
DISPLAY	The user's workstation
PRINT	An output file formatted for printing.

5.8 RUNTIME DEVICE AND FILE ASSIGNMENT

All action that relates to device and file allocation, file lookup, and control block generation is performed by the OPEN SVC. OPEN uses a number of parameters whose values must be specified in the user file block (UFB) or obtained through the GETPARM SVC. The UFB must be coded into the user's program or the user can take advantage of the UFBGEN macro (see Section 4.2).

The parameters solicited by the OPEN SVC through GETPARMS are used to assign device and file names to the internal file names within the user program. Some of these parameters are supplied only through the UFB when coded. Others are solicited using the GETPARM facility. Default values for solicited parameters may be stored in the UFB. GETPARM then enables the user to modify the default values; thus, the solicited GETPARM values override the UFB information.

5.9 DEFAULT FILE SPECIFICATIONS

The user can place default file specification information in the UFB at any time before a file is opened by using the UFBGEN macro. This facility should be used to minimize the amount of information to be entered by the user. Defaulted information includes the file name, library name, volume name, and the 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 to the size of the input file for the application. The size of a file is available in the UFB for any open input file. Thus, opening the input file first supplies the information needed to calculate default space requirements for the output file.

All information necessary to specify work files should be supplied in the UFB. Space requirements may be developed in the same manner as described for output files. File specification values, consisting of the file name and volume name, should be developed as follows:

- The volume location of the work file can be left blank in the UFB, to be supplied by means of a command language SET command.

- The library name for work files is ignored in the UFB before the file is opened. It is set to a special user work library associated with the logged on user by OPEN processing.

- The default file name should be formed using the characters "#" or "##" as a prefix to a maximum 4-character name which is unique to the program. OPEN appends a 4-character suffix to guarantee a unique temporary file name. Work files whose names begin with "#" are deleted when they are closed. Files whose names begin with "##" are not deleted by the operating system until the user returns to the command processor.

5.10 OPTIONS PRNAME

The OPTIONS parameter reference name should be used when a general list of parameter values is to be requested of the user. Care should be taken in supplying reasonable defaults values for these parameters.

5.11 ERROR HANDLING

There are several classes of errors that the operating system may encounter. Each requires a different response:

- Program exceptions in user programs are handled by the operating system's program check interrupt handler. This routine passes control to the VS DEBUGGER utility or to an exception handler (for example, the PCEXIT SVC) that the user program specifies.

- Program exceptions and other unrecoverable errors in system routines may result in a message displayed by the CANCEL SVC. If the cause can be traced to the user's program, CANCEL disallows continuation of program processing. (However, the program may continue running if CEXIT handlers have been defined.) If the problem is considered a probable system failure, CANCEL automatically initiates a system dump.

- If a system service receives invalid parameters, the routine should be able to detect this during initial validation of the parameters (before using them for further processing) and to return an error return code. Some services will issue cancels in this situation. Return codes are documented in the description of the system service (Chapter 3).

- There are three kinds of I/O errors: soft, hard, or logical file processing errors.

  - Soft errors signify that an I/O operation was successfully completed after retry by the I/O processor (IOP).

  - Hard errors signify failure of an I/O operation, including memory parity errors detected on an I/O operation.

  - Logical file processing errors do not reflect any errors that occur during an actual I/O operation.

Soft errors are passed to the CHECK SVC to be logged in a system error logging file, and are 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 makes reference to the file again by another Data Management System (DMS) 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 is entered for hard errors (if such a routine was provided). In the absence of an error routine, the user's program may be abnormally terminated at the discretion of the DMS routines, via issuance of a CANCEL SVC. Memory parity errors detected by I/O processors are logged in the same manner as other I/O errors.

The user's I/O error processing routine and end-of-data and invalid key condition routines are specified in the UFB. These routines are entered after interpretation of the I/O status word (IOSW) by the DMS 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.

Entrance to the above routines is from the unprivileged DMS routines as if the data management function had returned normally, but with the return address modified to be one of the addresses (the user's I/O error processing routine or end-of-data and invalid key condition routine) from the UFB. All register contents are restored except register 0, which is set to contain the normal return address from the function. Register 1 continues to address the UFB, which may be used to determine the nature of the unusual occurrence, as indicated in fields UFBFS1 and UFBFS2 of this block. 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 any other part of the user's program.

CHAPTER 6
VS OPERATING SYSTEM DESCRIPTION

6.1 INTRODUCTION

The VS operating system is a multiprogramming, time-sharing, virtual storage system that supports many users running programs at the same time. The operating system controls user programs and I/O devices and serves as a resource allocator. These activities include

- loading programs into memory
- scheduling work
- performing input and output operations on peripheral devices
- controlling print queues, file transfer queues, and background job queues

The operating system performs these functions in a manner that not only provides each user with all computer resources but also protects each user from the activities of other users on the system. This chapter briefly describes the basic functions of the operating system:

- How tasks are created (Tasks)
- How the operating system provides support for designing, implementing, testing, and executing tasks efficiently (System Support)
- How programs communicate with one another (Communication)
- How the operating system determines program execution priorities (Scheduling)
- How the operating system allocates resources, such as memory and I/O devices among programs requesting their use (Resource and Memory Management)
- How the operating system prevents unauthorized access to program and data (Security and Ring Memory Protection)
- How I/O subsystem routines communicate with peripheral devices (I/O Subsystem)
- The VS file structure

6.2 TASKS

The most basic unit of work in the VS operating system is the task. A task is the environment within which users and the system perform functions and run programs. The task environment is controlled by the Task Manager.

There are two categories of tasks: system tasks and user tasks. Among the system tasks are the printer task, the sharer, the file transfer manager, and the session manager. A user task is created whenever a user logs on, and is removed from the system when the user logs off.

Tasks do not perform all aspects of a function alone. To make the software more modular, tasks are often broken down into several subtasks. For example, when a user logs on the system, a task is created. If the user creates another task, that task becomes a subtask.

The following operating system tasks are always present:

- Task Manager: Creates user tasks.

- System task: Performs operator functions (for example, task statuses) and manages queue print procedure.

- Printer task: Manages spooling.

- Sharer task: Manages shared file access.

6.2.1 Task States

The VS operating system creates system and user tasks dynamically. During the IPL procedure, the operating system generates control blocks which are used to maintain information regarding the many tasks active within the system.

From the time a task is created until the task is destroyed, it exists in one of four task states. Tasks move among these states as they are created, begin execution, and finally complete their functions. Table 6-1 describes the four task states.

Table 6-1. Task States

State	Description	Example
Active	Task is currently running	Executing on the CPU
Runnable	Task is ready to be executed on the CPU	On one of the internal priority queues
Waiting (blocked)	Task waiting for a resource	Waiting for disk access, or blocked on a semaphore
Suspended	Task is stopped	Occurs when there is a task crash

6.2.2 Task Scheduling

One of the functions of the operating system in a multiprogramming environment is to apportion central processor time for runnable tasks. This function is handled by the scheduler and the dispatcher.

The scheduler determines the order in which runnable tasks will be given control of the processor (for more information on the scheduler, refer to Section 6.5). When switching between tasks, the dispatcher must save the state of a task which has been blocked, choose the next task to run, and restore the running state of the chosen task. Runnable tasks are given control of the CPU by the dispatcher in the priority order determined by the scheduler. The operating system maintains a data structure known as the Ready List. The Ready List contains descriptions of tasks which are ready to be executed by the CPU. The dispatcher determines which task on the Ready List should be executed next and the maximum amount of time it should execute.

While a task is executing, various types of events can cause the task to suddenly stop executing. For example, a task may need to wait for the completion of an I/O operation, the expiration of a timing interval, or the receipt of a message from another task. When a task stops executing because it is waiting for an event, it is said to be blocked. A task remains in this blocked state until the event for which the task is waiting occurs. A task may also be interrupted by an unsolicited interrupt from a workstation, printer, or telecommunications device.

To track the status of a task, each task has a context and some task control blocks associated with it. The task context is the information that specifies the complete status of a task (for example, the registers, the point in the program when removed from the CPU, instruction pointers, and memory locations). The task context is saved in a task control block which also contains other information such as the task ID, priority, and the current state of the task. When a task is interrupted, the context information is saved to allow the operating system to resume execution of that task at some later point without error.

6.2.3 Event Scheduling

An event is a significant occurrence to a task such as an I/O interrupt, the arrival of a message at a task's mailbox, or the occurrence of an exception condition. To control task execution, the operating system uses semaphores. Semaphores act like gates into critical areas of software to protect shared data or I/O.

Section 6.4.1 contains more information about semaphores.

6.2.4 System Task Queue Verification Routine

At IPL time, the System Task automatically invokes the queue file verification routine, @QUEVER@, if an existing queue file is on the system disk. This task runs in the background and verifies the correctness of the entries within the queue file. Upon completion of the routine, the task displays the following message on the operator screen:

```
Msg From QVR: Queue verification routine complete.
```

If a program check occurs, the @QUEVER@ routine is automatically invoked in order to rebuild the system queue file. The @QUEVER@ routine recovers as much good data as possible from the corrupt version. During appropriate times in the process, the routine displays the following messages on the operator screen:

```
Msg from QVR: Cancel condition in SYSTSK - Rebuilding System
Queue.

Msg from QVR: Queue verification error - Bad entries in Q
file.

Msg from QVR: Queue verification routine complete.
```

The error handling logic within the System Task enables the module to continue running in spite of any arising queue corruption.

6.3 SYSTEM SUPPORT

System support includes programs and tasks that enable a user to design, implement, test, and execute tasks efficiently. The programs and tasks that support software development are called utilities. Unlike the system services, these utilities are not part of the basic operating system. However, these utilities are invaluable in making the VS system easy to use.

The VS provides the following categories of support functions:

- Language Translators
- Program Editing and Linking
- Debugging
- System Configuration
- Performance Monitoring

6.3.1 Language Translators

To support the development of new applications, the VS operating system provides two types of language translators, an assembler and various compilers. The assembler produces machine code from a very low level input language. The programmer can directly access the VS machine instructions through assembly language.

System programs known as compilers accept progam text written in a high-level language (such as COBOL or PL/I) and translate this text into actual machine instructions. The compilers and assembler for all VS-supported languages automatically check for syntax errors on all source code that progammers enter into the system. To help programmers easily identify and eliminate program bugs, the VS operating system displays clear diagnostic error messages.

6.3.2 Programming Editing and Linking

The VS EDITOR utility fully integrates all functions needed to create, edit, compile and run programs written in any language supported on the VS. The VS LINKER utility provides the capability to link several individually compiled program modules, which may be written in different languages, into a single executable load module.

6.3.3 Debugging

Debugging is the process of identifying and correcting errors in the program code to ensure that the program is logically correct. The VS operating system provides an Interactive Symbolic Debugger, a powerful facility that allows programmers to examine and modify data in memory, referring to memory location names rather than absolute addresses.

The Debugger also includes an optional trap handler that allows you to set various break points in the program. Programmers can then step through program execution, stopping at predetermined addresses or instructions to examine data structures and register contents, and modify data where appropriate.

6.3.4 System Configuration

Because of variations in different installation environments, it is often necessary to tailor the system for a particular installation. The VS Operating System GENEDIT utility allows you to specify hardware and operating system options for your system by enabling you to create or edit a configuration file. You can run GENEDIT at any time after the system is IPLed, even while other users are on the system.

Because the configuration information is stored in a file, you can have several files that correspond to different configurations on the system at one time. During IPL, a screen will request the name of the configuration file to use.

6.3.5 Performance Monitoring

Good performance monitoring is a critical factor in any multiprogramming environment. On the VS system, the System Activity Monitor (SAM) utility (if is available on your system) collects data on the use of three primary resources: the CPU, disks, and main memory. With this information, users can take steps to improve a system's performance, such as reprioritizing user programs, redistributing disk files, or adding devices. SAM also monitors the performance of individual tasks and programs. This individual monitoring provides a significant tool for analyzing and improving program efficiency.

SAM is a menu-driven utility that operates in both Interactive and Background modes. In Interactive mode, a user can investigate a performance problem as it occurs, allowing for an immediate adjustment to the system. In Background mode, SAM provides extended, concurrent monitoring of multiple activities.

To monitor paging activity, users can evaluate the paging rate by using the Control Interactive Tasks (Paging and I/O) screen of Operator mode or the SAM utility. For more information, refer to the VS System Operator's Guide and the VS System Activity Monitor (SAM) Reference.

To monitor page pools, users can access the POOLSTAT utility. The VS system monitors the use of page pools and issues warnings when a page pool nears capacity. Through the POOLSTAT utility, users can view page pool statistics at any time. Current and peak usage statistics can help users determine if the VS system's current page pool capacity is adequate for the paging requirement of the tasks assigned to the page pool.

To display statistics that the Sharer collects as it processes user requests, use the SHRSTAT utility. For more information on POOLSTAT, and SHRSTAT, refer to the VS System Utilities Reference.

6.4 COMMUNICATION AND SYNCHRONIZATION

To coordinate system operation, tasks communicate with one another in the VS operating system environment. Tasks communicate with each other by exchanging both commands and data. This information can be transmitted through main memory or secondary storage.

When data is transferred, the receiving task must be ready for it, and the newly transferred data cannot be allowed to invalidate previously transmitted data. To ensure the coordination of these specific events in the transmission process, the VS Operating System uses various synchronization techniques: semaphores, the Intertask Messaging (ITM) facility, and the User Synchronization facility. The following sections describe these techniques.

6.4.1 Semaphore

The semaphore is a general synchronization technique that is not available to user-level code. Semaphores act like gates into critical areas of software to protect shared data or I/O. All semaphores are in protected memory locations.

A semaphore permits only a single task to access shared data at any given time. All other tasks are locked out until the first task unlocks the shared data. Initially, the gate is open, but when the first task enters the restricted area, it automatically closes the gate to lock out any other tasks. Once the task enters the critical section, it may access the shared data and no other task is permitted access until the current task is finished.

User programs do not have the ability to modify semaphores directly. However, they can use the CHECK routine to wait for notification that the event has occurred and that the gate is open.

6.4.2 Intertask Messaging (ITM)

Blocks of data and commands that are communicated from one task to another are called messages. The Intertask Messaging (ITM) facility performs the following services:

- Creates and destroys a message receipt port for communicating with other tasks

- Transmits a message to another task

- Checks for an event occurrence

Communication between tasks is provided through the CREATE, XMIT and CHECK SVCs. To receive intertask messages, an application program must use the CREATE SVC to create a message port. A message port is a buffer that is used to hold messages and resides in memory. This message buffer can be either resident or nonresident.

Associated with the message port is a 4-character port name. The XMIT SVC is used to send messages to the named port. XMIT searches the message port chain, looking for the specified port name. If it finds it, XMIT copies the message to the end of the message port. If no space is available, XMIT either waits for space to become available, or returns to the caller if NOWAIT is specified.

The task owning the port receives a message by issuing CHECK MESSAGE. CHECK waits for a message if none is available (or optionally, the user may use the NOWAIT or multiple event option). When the message is received, CHECK copies it to the caller's modifiable data area. To remove the port, the receiving task uses the DESTROY SVC.

6.4.3 User Synchronization Facility

The VS Operating System supports a user synchronization facility that allows you to control resources, such as access to a database, so that only one task at a time is in control of the resource.

The user synchronization facility provides a fast, simple synchronization technique for controlling access to shared data in user-level code. System services that can be called from a user program allow a user to create, delete and use a synchronization object to coordinate the access to shared data. The synchronization object is used most often for resource control -- that is, to update a database or access a particular piece of code. However, it may be used to satisfy other application needs as well.

The operating system has no knowledge of what is being controlled by the synchronization object; it only provides single user access to the synchronization object. It is up to the user to ensure that all users of the resource go through the synchronization facility. Note that the operating system does not clean up the user synchronization facility control blocks if the synchronization object creator cancels itself or is cancelled before explicitly destroying the synchronization object.

For more information on the user synchronization facility, refer to Section 1.2.4. For a description of the associated calling sequences, refer to System Services in Part II of this manual.

6.5 SCHEDULING

The scheduler determines the order in which runnable tasks will be given control of the processor. Control passes to the scheduler under the following conditions:

- When a task must wait for an event to occur
- When a task becomes ready as a result of the occurrence of an event
- When an interrupt has occurred, and preliminary processing of the interrupt is complete, (e.g., on exit from the I/O interrupt service routine or clock interrupt service routine)
- When a System Must Complete state exists, and the user presses the HELP key

When a task is ready to execute (i.e., not blocked waiting for resources), the scheduler determines the priority at which it should be dispatched. The dispatcher then determines the highest priority ready-to-run task, and sets a timer expiration value to ensure that this task does not monopolize the processor at the expense of other tasks.

6.5.1 Categories of Tasks

Some tasks are usually considered more important than others. As a result, the scheduler must use a priority system to allocate time. Higher priority tasks can access the CPU more frequently. Lower priority tasks access the CPU less frequently.

Some tasks, known as I/O-bound tasks, require short bursts of CPU time intermixed with many long waits for I/O completions. Tasks tend to become I/O-bound when you run programs such as WP, EDITOR, DISPLAY, COPY, PRINT, and some applications programs.

Other tasks, known as CPU-bound tasks, require long periods of CPU time intermixed with a few long waits for I/O completions. Tasks become CPU-bound when you run programs such as the compilers, the Linker, and specific commercial applications that require extensive CPU resources.

In addition, tasks can go through phases in which they become I/O-bound and phases in which they become CPU-bound, while running the same program.

It is generally desirable to allow I/O-bound tasks, which have light CPU needs, to be given priority over CPU-bound tasks. However, once CPU-bound tasks gain access to the CPU, they can retain it for a longer period of time.

There are two categories of tasks: system tasks and user tasks. System tasks exist as part of the operating system to serve various user needs. These system tasks perform functions such as

- soliciting and accepting user logons
- managing telecommunications links
- running background jobs
- providing some operator functions
- managing virtual and physical storage
- printing DP and WP documents.

When a user task makes a request to a system task, the user task must usually wait until the system task completes the service. Because there are frequent requests for system tasks, effective scheduling of these tasks is a critical performance issue.

6.5.2 Scheduling Formula

System tasks get the highest priority, followed by user tasks. Within the system task group, tasks which service user tasks, such as the pager, the system task, and the Task Manager, are given higher priority than tasks which run independently, such as printer tasks.

The system maintains sixteen dispatch queues for tasks. Systems tasks are placed into queues 0-7. User tasks are placed into queues 8-15.

To improve user-response time, the VS Operating System uses a scheduling formula that allows users to assign priorities to user programs (users cannot assign priorities to system tasks). This scheduling formula is based on a set of priority levels.

Within the user task group, users can set the following priorities: high, medium-high, medium-low, and low. Within each priority level, I/O-bound tasks are given higher priority than CPU-bound tasks. This means that the user retains good response time while editing a program source file, even if several compiles or links are running.

This scheduling formula allows the operating system to make better distinctions between I/O-bound tasks and CPU-bound tasks within each priority group. The option to set priorities for individual user programs allows users to tune the scheduling according to their business needs. This tuning ability gives users a significant advantage. They can receive quick turnaround on the more critical jobs and thus respond more effectively to their business objectives.

To specify program priorities, the user must select the program priorities option in the GENEDIT utility and then IPL the system using that configuration file. To actually set or change a priority for an individual program, the user must run the SECURITY utility. By default, user foreground tasks are assigned medium-high priority, and user background tasks are assigned medium-low priority. For a complete description on how to use these options in GENEDIT and SECURITY, refer to the VS System Administrator's Reference.

6.6 MEMORY MANAGEMENT

Memory management includes a combination of hardware and software that controls the allocation and use of physical memory for VS systems.

The VS memory management scheme is designed to:

- Provide a large address space for instructions and data
- Provide efficient sharing of instructions and data
- Contribute to software reliability

In the VS multiprogramming environment, the code and data required by several tasks may reside in physical memory at the same time. Therefore, one of the functions of memory management is to provide memory protection and to control access to memory (refer to Ring Memory Protection in Section 6.7). To accommodate many VS users with simultaneous access to main memory, a virtual addressing scheme provides programs with a much larger address space than the actual physical memory supported by the hardware configuration.

Before virtual addresses can be used to access instructions and data, they must be translated to physical addresses. Memory management maintains page tables that keep track of where each 2-KB virtual page is located in physical memory. Memory management uses this mapping information to translate virtual addresses to physical addresses. This process is called address translation. For a complete description of address translation, refer to the VS Principles of Operation.

6.6.1 Virtual Address Space

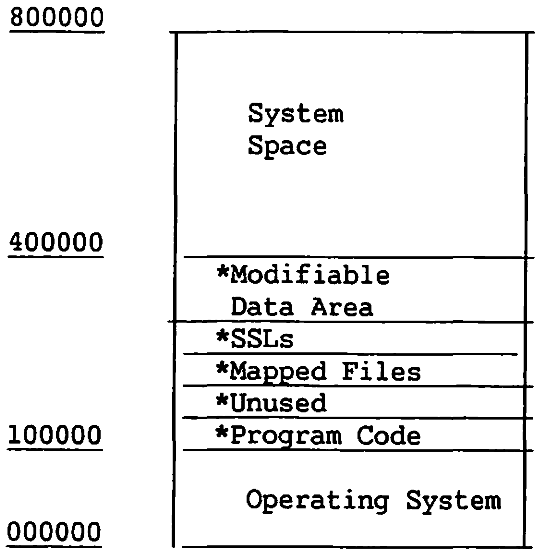
The VS Operating System uses the memory management functions described in this section to provide each user with a potential 8 million bytes of virtual address space on 16-MB systems.

Depending upon the hardware configuration, a VS system may have from 1 MB to 16 MB of physical memory. Physical memory is limited to a maximum of 16 MB on both the VS300 and the VS85/90/100, 4 MB on the VS65, and 2 MB on the VS15/25/45.

Through the use of on-line disk storage and an addressing mechanism which is part of the operating system, the amount of memory available to the user is extended beyond the amount of physical memory actually installed on the system. The availability of 24-bit virtual addressing allows the programmer to write a program which could exceed the amount of physical memory on the system. For example, with 16-MB virtual addressing support, the user can address up to 8128 KB of user address space.

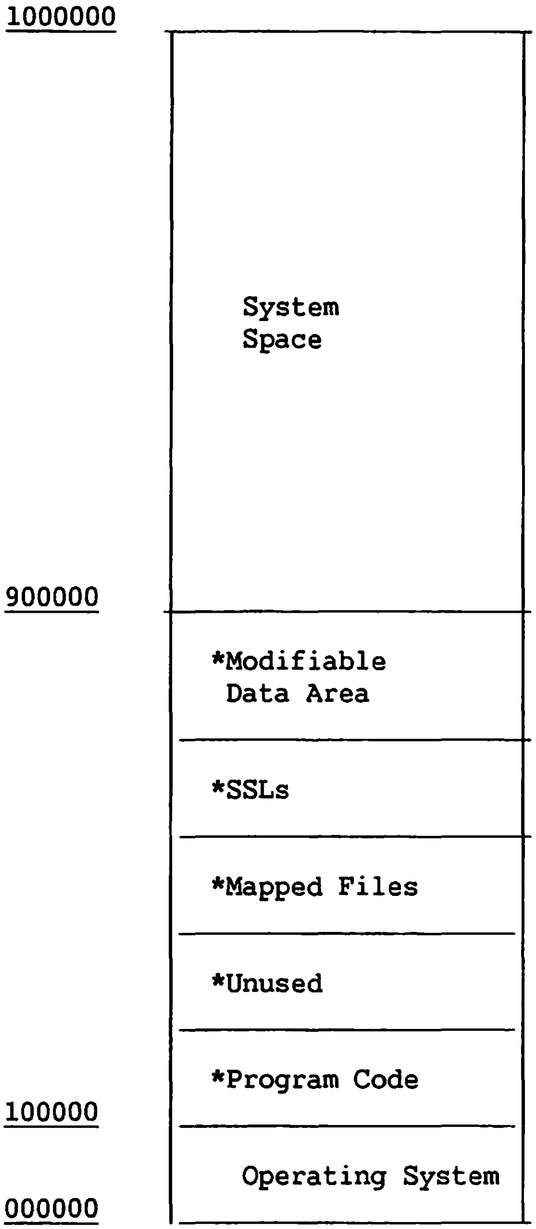
The user is not aware of the specific functions involved in providing virtual memory and need not be concerned when programming an application for the system. However, an understanding of the mechanism involved aids the user in understanding the protection mechanism and the functioning of the operating system.

Figures 6-1 and 6-2 illustrate how address space is allocated on 8-MB and 16-MB VS Operating Systems, respectively.



* User Address Space (variable size)

Figure 6-1. VS 8-MB Address Space Allocation



* User Address Space (variable size)

Figure 6-2. VS 16-MB Address Space Allocation

The organization of internal memory allows programmers to run larger programs than they could on previous releases of the operating system. In addition, users now have more flexibility in defining their virtual address space.

Users can modify the size of various areas in their virtual address space, which gives them greater flexibility in defining the appropriate address space for an individual program. Each user's virtual address space is divided into the following areas:

- User modifiable data area

- Optional address space area (can be used for shared subroutine libraries or mapped files)

- User program code area

System administrators can specify the size of each user's modifiable data area using the GENEDIT and SECURITY utilities. The default size for the modifiable data area can be specified through the GENEDIT utility. This size can be overridden by using the SECURITY utility. Both the GENEDIT and the SECURITY utilities are described in the VS System Administrator's Reference.

Users have more than 1 MB of memory address space for code, and more than 1 MB of memory address space for data. The VS system provides each user with up to 3008 KB of contiguous logical address space on 8-MB systems and up to 8128 KB of contiguous, logical address space on 16-MB systems. When the size of the modifiable data area is pre-allocated, this area remains dedicated to the program stack, buffer, and static data, even if it is not completely utilized. All remaining space in the user's address area is available for program code. If there is additional space in the user address area after the user program code is loaded, this remaining space can be used for shared subroutine libraries, or for mapped files.

In essence, users can mix and match code and data space. By manipulating data and program code space, address space can be used more effectively. For example, users can assign 1/2 MB of memory to data and 2 1/2 MB to code, or assign 2 MB for code and 1 MB for data, depending on the needs of the application.

6.6.2 Relationship of Virtual Memory to Physical Memory

The operating system and the microcode manage physical memory in such a way that each user appears to have a large, contiguous area of memory available for programs and data. The memory that the user program addresses is referred to as virtual memory, and constitutes the user's individual virtual address space; the actual main memory is called physical memory.

The virtual address space of all users of a system collectively exceeds the amount of physical memory available on the system. So memory management provides the mechanism to map the part of each user's address space into physical memory. The VS Operating System controls the addressing mechanism (i.e., page tables) that map virtual addresses into physical memory addresses. Parts of a user's virtual address space that are temporarily inactive are mapped onto external disks by the operating system.

6.6.3 Regions

Virtual address space is divided into units known as regions. A region is a contiguous portion of a task's virtual address space that begins on a page boundary and contains a variable number of pages. The number of regions can vary, although there is a maximum number of 64 regions per virtual address space.

6.6.4 Pages, Page Faults, and Address Translation

A region is mapped to coincide block for block with a paging file (either a program or a data file) on disk. When the operating system runs a program, it retrieves the program or data from the disk and loads it into main memory. (System tasks are also programs and are loaded in the same way.)

Programs and data files are stored externally in 2-KB segments called pages. Physical memory is partitioned into 2-KB areas to receive these pages. These areas of contiguous memory are called page frames. The process of loading pages into and out of main memory is called "paging." Certain page frames are dedicated to operating system routines and data which must be resident in memory at all times. Other page frames may contain parts of any other program, parts of the operating system, or data.

When a program is submitted to the operating system to be run, the operating system loads pages of the program as they are needed. The operating system attempts to maintain as many pages of the program as it can (depending upon system processing demands) into main memory. When loading a program, the operating system records in a task's page tables the number of the physical page frame that receives each page. Using a task's page tables, the CP translates virtual addresses into physical memory addresses from the time a program is loaded into memory until it completes running and exits.

An executing program may address any one of sixty-four contiguous regions. The high-order bits of the virtual address, which specify the virtual page number, are used to select the particular region (Figure 6-3).

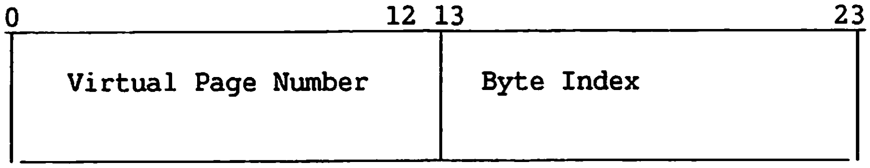


Figure 6-3. The 24-bit Virtual Address

The hardware uses bits 0-12 (Figure 6-3) to select a region (and hence a page table), each item of which addresses a page frame in physical main memory. The same bits of the original address are used to select table items within a page table, and thereby 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.

Although table items may be addressed within a page table, not all may fit into the available physical memory. 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, performs a program check interruption known as a page fault. Then, it supplies to the program interrupt service routine the region and page numbers of the missing page.

The pager task, initiated by the interrupt service routine, 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. It then 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. It is then reactivated after the page-in operation is completed and the paging task has updated the page table to allow normal addressability of the page.

If the page that is paged-out had been modified, that page is placed into a paging area on the disk. This paging file will either be assigned by the system, or to a page pool that was previously defined with the DISKINIT utility.

When a page pool is allocated, an area on the disk is specified where modified pages are placed if main memory is needed for other tasks. The page pool is a temporary location for pages that are being modified during the processing of a task or program.

Specifying the location, size, and commitment ratio for the page pool is important for efficient system performance. For detailed information on these topics, refer to the VS System Utilities Reference.

6.6.5 User Program Efficiency and Paging

Designing a program for a virtual memory environment requires a new outlook toward program and data organization. Although one is freed from the task of managing small physical storage by overlay or other manual segmentation techniques, the user cannot ignore the issue of program organization. A major aim of the programmer should be to increase the reference locality of his program's code. That is, the programmer should avoid referencing many pages of code or data within a short span of program execution. This reduces the likelihood of many page faults occurring.

6.7 RING MEMORY PROTECTION

The VS operating system supports a hierarchical, "ring" memory protection scheme to provide greater internal security. Ring memory allows the operating system to internally implement several levels of protection. These levels of protection are designed to control access to memory, and to further contribute to the integrity of the system.

6.7.1 Process Levels

Under the ring memory scheme, access to memory and privileged instructions is controlled by a series of process levels, or rings. Ring assignment ranges from 0 to 7 as described in Table 6-2.

Table 6-2. Internal Memory Process Levels

Process Level	Meaning	Use
0	Corresponds to current user state	For all program execution
1-6	Expanding levels of protection	For system support facilities
7	Corresponds to current privileged state	For OS kernel

Process level 0 is reserved for all user program execution. The intermediate process levels, 1 through 6, define increasing levels of privilege. Any location accessible to a less privileged process level is also accessible to all more privileged levels. Process levels in the intermediate range permit tasks to access level 0 or the next highest level in the range. Currently, level 1 is used for system facilities, such as the Command Processor and the Debugger. Level 3 is reserved for DMS routines.

Memory protection is on a regionwide basis. Each region of virtual address space has a minimum read level and a minimum write level (two numbers associated with every region). Access to a region is controlled by the region's protection levels and the current process level which is recorded in the PCW (Program Control Word) privilege bit. Bits 61-63 of the PCW contain the process level (for example, that level of privilege at which the user application or service is running). To determine the access rights of a piece of code, the system microcode compares the process level value in the PCW to the Read and Write levels in the region node table (RNT) for the logical address that the program is trying to access. For more information concerning the region node table, refer to the VS Principles of Operation.

6.7.2 System Stacks

To further enhance system security, the VS ring memory protection scheme supports a separate system stack for every process level. Each stack is protected from being accessed by a lower process level. A stack maps into a paging file that is opened exclusively by the task. When the process level of a task changes, the system microcode activates the new level's stack.

Also, each stack supports a buffer (heap) area. The GETHEAP and FREEHEAP SVCs allocate and deallocate space in the currently active stack, but may be directed to the stack of a lower process level. The buffer area is used for protected data by the system services.

For more information on system stacks, refer to the VS Principles of Operation.

6.7.3 JSI-type System Services

The ring memory protection scheme supports JSI-type system services, which are system instructions with a process level of 1 or higher. JSI-type system services check for protection, as opposed to SVC's, which do not. During task initialization, shared files containing JSI-type system services are mapped into every task's address space.

JSI-type system services have the following characteristics:

- Can be called directly by any language

- Allow entry to protected address space (ring level 1 - 7) using JSI (Jump To Subroutine) instructions

- Can have a static area and can allocate buffer storage. Both the static and buffer areas are read-protected and write-protected from user code

Access to JSI-type system services is controlled by system microcode, which tests for the process level required for entry. If a task has a lower process level number, access to a JSI-type system service is not permitted.

For more information on JSI-type system services, refer to Chapters 2 and 3.

6.8 THE I/O SUBSYSTEM

The I/O subsystem allows the user to perform I/O operations to a variety of devices (for example, magnetic tapes, disk files, telecommunications devices, printers). The I/O subsystem routines can be divided into three areas which interact to effectively communicate with the peripheral devices. These three areas include

- The I/O initiation routines, including the XIO SVC, LOADCODE SVC, and the system physical start I/O routine
- The I/O completion routines, including the I/O interrupt handler, together with CHECK SVC, and HALTIO SVC
- The asynchronous system I/O monitoring task

The XIO SVC is the user-accessible I/O initiation routine which allows the user to read or write data to a device on the system. The system STARTIO routine actually issues the SIO instruction to initiate the I/O request.

The LOADCODE SVC is used to load microcode to programmable devices and IOPs. This routine can also be called by the user and invokes the system STARTIO routine. It can be automatically invoked through the CHECK SVC when a device or IOP is found to be lacking microcode that is needed to successfully complete an I/O operation.

The CHECK SVC is a routine used to self-suspend the user's task while awaiting the I/O completion. This routine uses the OFB address as input, so that it can locate the appropriate semaphore address to wait upon. I/O is synchronized by semaphores shared with I/O completion routines. The HALTIO SVC cancels an I/O operation initiated by XIO.

Refer to Chapter 4 for more information on these SVCs.

6.9 VS FILE STRUCTURE

The operating system maintains a 2-level file structure (library name and file name) on a disk volume, stored in a volume table of contents (VTOC) which maintains information for the location of files within libraries.

Disk volumes are divided into 2048-byte blocks, numbered from zero. Files on a volume are written in one or more contiguous areas, called extents. Each extent spans one or more consecutively numbered blocks. The presence of a file is indicated in the VTOC, located through the volume label.

6.9.1 Volume Label

The volume label occupies block 0 of a disk volume. It contains the name of the volume (the volume serial number), the location of the volume's table of contents, and other descriptive information defining the size and physical organization of the volume.

6.9.2 Extent Organization

Each block on a volume, with the exception of the first block and the blocks containing the VTOC, is part of an extent. Each extent is either part of the available space record in the volume table of contents or part of the file space which is recorded in file descriptor records (FDRs), also in the VTOC. FDRs contain information specific to a particular file.

The first FDR for a file is referred to as FDR1. All data management and historical system information is contained in the FDR1 record. For files which reside on independent volumes (not part of a volume set), the FDR1 record contains the first three extents of the file. Additional extents (up to 255) occupied by a file are described by FDR2 entries.

For files which reside on volume sets, the FDR1 record contains pointers to the FDR2 and FDR3 records. FDR2 records contain extent information; FDR3 records contain segment information. The information in these records is used to locate the volume which contains a particular file-relative block number. FDR3 records contain segment descriptors, consisting of a volume ID number, a segment sequence number, and a starting block number.

When a file is initially allocated space, an attempt is made to acquire a single extent of sufficient size on the volume. If such an extent is not available, up to 255 extents may be allocated and are described in the FDR1 and FDR2s. For multivolume file support, the first nine extents are described in the first FDR2, and the remaining extents are described in subsequent FDR2s and FDR3s. For volume sets, the extent allocation may be unlimited.

The system administrator can specify the maximum number of extents that are allocated to a file upon creation and upon extention by using the DISKINIT utility. Refer to the VS System Utilities Reference for more information on file extents and volume sets.

NOTE

For files which reside on volume sets, the extents allocated at creation time must fit on one member of the set. These extents will not span multiple volumes.

6.9.3 Volume Table of Contents

The volume table of contents (VTOC) is structured in the following way:

- The first block of the VTOC is an available space block. When searching for space on the volume to store a new file, the operating system searches this block first (followed by its chain of unused blocks) for a sufficiently large area of space.

- The second block of the VTOC is an index block. This first-level index block, which describes a user library, may be chained to additional first-level index blocks. The first-level index block contains a pointer to second-level index blocks.

- The third block of the VTOC is a second-level index block which describes a file within a library. This block is the first in a chain of additional second-level index blocks.

- The fourth block of the VTOC is a block containing File Descriptor Records (FDRs).

- The fifth and additional blocks of the VTOC are used for available space blocks, index blocks, or file descriptor blocks.

User programs create files by invoking the OPEN SVC and can create any file in any library providing that the combination of file and library do not already exist. Files may be removed from a disk by invoking the SCRATCH SVC. Any user can delete any file in any library providing the security constraints are obeyed. Deleting a file consists of removing the FDR(s) which describe the file and freeing the space which it contained. If a file is the last one in its library, the library must also be removed by deleting the corresponding index block. A file name may be changed by updating the FDR1 to reflect the new name. File and library names are obtained by invoking the READVTOC SVC.

APPENDIX A
PROGRAM FILE STRUCTURE AND PROCESSING

A.1 THE PROGRAM FILE STRUCTURE

A program object file is partitioned into blocks of information. Each block represents common information that is needed either when the program is running or when the program is processed by the Linker. As shown in Figure A-1, a program consists of three blocks: the run block, the symbolic block, and the linkage block. This is standard for the two possible object file formats that can be processed by the Linker. Object format Version 0 is the format used for Release 6.00 series of the VS Operating System. This format is documented in the Section A.2. Object format Version 1 is the native format for Release 7.00 series of the VS Operating System. It is described in Section A.3.

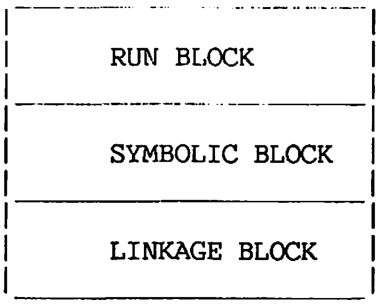


Figure A-1. Program File Structure
Object Versions 0 and 1

A.2 OBJECT FILE FORMAT FOR RELEASE 6.00 SERIES

Object file format 0 for Release 6.00 series of the VS Operating. System is described in this section. Refer to Section A.3 for the description of object file format 1 used for the Release 7.00 series of the VS Operating System.

A.2.1 The Run Block, Version 0

The run block contains the information needed by the system to run a program. It is 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 starts. The first location of the run block has a virtual address of X'100000' (1024k).

				Bytes	NOTES
RUN BLOCK	CODE AND PROLOG BLOCK	PROLOG BLOCK	LENGTH OF CODE AND PROLOG BLOCK	4 bytes	1
			ENTRY POINT ADDRESS	4 bytes	2
		CODE BLOCK	CODE SECTION	(variable)	3, 4
	LENGTHS BLOCK		LENGTH OF STATIC BLOCK OBJECT TIME	4 bytes	5
			LENGTH OF STATIC BLOCK RUN TIME	4 bytes	6
			RESERVED MUST BE ZERO	4 bytes	
			RESERVED MUST BE ZERO	4 bytes	
	STATIC BLOCK		SEE FIGURE A-3	(variable)	7

Figure A-2. The Run Block, Version 0

Notes on Figure A-2:

1. The length of the code and prolog block is used to find the start of the static and lengths blocks.

2. The entry point address is the address to which control is passed when the program is started. It is 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. The code block may contain any number of code sections. This block contains all of the executable instructions in the program and may contain unmodified data. It is composed of any number of sections, where a section is a contiguous area of code that can be moved as a whole by the Linker program. There is no requirement for a particular order of the sections within the code block.

4. The code section is a block externally identified by its name. It is an independent contiguous area of code supplied by the language translator. The first location is on a doubleword boundary, and the length is divisible by eight. All address constants that are resolvable are resolved so that the program can be run without changing any locations in the section.

5. The length of the static block in bytes reflects the length of data in the static area at object time. If the length is not divisible by 4, up to three bytes of slack are 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.

6. This is the length of the static area in bytes at run time.

7. The static block contains sections of initial value records. There can be any number of static sections in this block, including zero.

The Static Block, Version 0

The static block contains initial value records that are to be processed by the program startup facility in the operating system. These records 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 are resolved by Linker or Translator programs as if they were in a code section. Address constants that address locations within the static sections are resolved as if the start of the static block were location zero.

					NOTES
STATIC BLOCK	STATIC SECTION	INITIAL VALUE RECORD	LENGTH OF DATA IN DATA FIELD	1 byte	1, 2, 3, 4
			RECORD TYPE	4 bits	5
			RUN TIME DISPLACEMENT	2 bytes	6
			DATA FIELD SEE FIGURE A-4	1-256 bytes	7

Figure A-3. The Static Block, Version 0

Notes on Figure A-3:

1. The static block may contain any number of static sections, including zero.

2. 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 have initial values, there are no records for that section and the object time length is zero. 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 is used. Locations in the object code file are referred to as object time locations or are specified by their object time address. Locations that are used during running of a program are referred to as run time locations or are specified by their run time addresses. Because the code block is without change at run time, this distinction is normally not made for any locations in 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.

3. Initial value records specify locations that are to have initial values in the named static section. The initial value records also specify the values the program startup mechanism is to assign to these locations. There are five types of initial value records:

   - The origin record, which specifies how far from the start of the expanded run time static area this section starts.

   - The value record, which specifies the value to be placed in the static area.

   - The relocation record, which specifies that program startup is to supply the address of a run time location in a static area.

   - 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.

   - The compressed record, which specifies the compressed value to be expanded and placed in the static area.

4. Length of data within the data field in this record minus one. This field is not used for the compressed record type.

5. The record type:

    - 0 = Value
    - 1 = Origin
    - 2 = Relocation
    - 4 = Repeated
    - 8 = Compressed

6. Run time displacement. This field has two interpretations. For origin records (record type 1), this field indicates the displacement from the start of the static area of this static section. For all other record types, this field indicates the run time displacement from the start of the static section of the record's data.

7. Data field. The length and format of this field vary depending on the record type.

The Data Field for:

Value Record: NOTES

DATA FIELD	DATA TO BE MOVED TO THE STATIC SECTION	1 - 256 bytes	1

Origin Record:

DATA FIELD	DUMMY DATA	1 byte	

Relocation Record:

DATA FIELD	RESERVED; MUST BE ZERO	4 bits	
	LENGTH OF TARGET ADDRESS CONSTANT	1 bit	2
	DIRECTION OF RELOCATION	1 bit	3
	DO NOT RELOCATE FLAG	1 bit	4
	INITIAL VALUE OF THE ADDRESS CONSTANT	25 bits	5

Repeated Record:

DATA FIELD	REPETITION FACTOR	2 bytes	6
	DATA TO BE REPEATED WITHIN THE STATIC AREA	1 - 256 bytes	7

Compressed Record:

DATA FIELD	LENGTH OF COMPRESSED DATA	2 bytes	8
	COMPRESSED DATA	1 - 2048 bytes	9

Figure A-4. The Data Field, Version 0

Notes on Figure A-4:

1. This data is moved unchanged to the run time static area.

2. Length of target address constant

   - 0 = three bytes
   - 1 = four bytes

3. Direction of relocation:

   - 0 = Positive
   - 1 = Negative

4. The do-not-relocate flag is used if the address is unresolved, if the address referenced a code location, or if this is an R type address constant.

   - 0 = Relocate the address constant.
   - 1 = Move the address constant to the specified location but do not relocate with respect to the static area.

5. Initial value of the address constant. If the target is three bytes long, only the last three bytes are moved to the target area and the high-order bit is ignored. If the target is four bytes long, the high-order bit is propagated through the seven remaining bits of the high-order target byte.

6. The repetition factor may have a value from 2 - 32767.

7. The data to be repeated within the static area.

8. The length of the compressed data may have a value from 1 - 2048.

9. The compressed data is expanded before being moved to the static area.

A.2.2 The Symbolic Block, Version 0

The symbolic block is a pool of information used by the system's debugging program. The block is partitioned into a length field and any number of section-related blocks.

```
                                                                         NOTES
 ___________________________________________________________
|          | LENGTH OF SYMBOLIC BLOCK                        |
|          | FULLWORD ALIGNED________________________________|  4 bytes   1, 2
|          |          | LENGTH IN BYTES OF                   |
|          |          | DATA IN THIS SECTION_________________|  4 bytes   3, 4
|          |          | EXTERNAL NAME OF                     |
|          |          | CORRESPONDING CODE SECTION___________|  8 bytes   5
|          |          | OBJ TIME LENGTH OF                   |
|          |          | SECTION IN RUN BLOCK                 |
|          |          | DOUBLEWORD ALIGNED___________________|  4 bytes   6
|          |          |          |                           |
|          |          | SOURCE   | FILE NAME_________________|  8 bytes   7
|          |          | FILE     |                           |
|          |          | LOCATION | LIBRARY NAME______________|  8 bytes
|          |          |          |                           |
|          |          |__________| VOLUME NAME_______________|  6 bytes
|          |          |          |                           |
|          | SYMBOLIC | LISTING  | FILE NAME_________________|  8 bytes   8
| SYMBOLIC | SECTION  | FILE     |                           |
| SECTION  | AREA     | LOCATION | LIBRARY NAME______________|  8 bytes
|          |          |          |                           |
|          |          |__________| VOLUME NAME_______________|  6 bytes
|          |          |          |                           |
|          |          |EXTERNAL  | LENGTH OF POOL____________|  4 bytes   9, 10
|          |          |REFERENCE | ANY # OF | EXTERNAL       |
|          |          |POOL      | EXTERNAL | NAME___________|  8 bytes  11, 12
|          |          |          | NAME     | CODE SEC DIS-  |
|          |          |__________| ENTRIES__| PLACEMENT______|  4 bytes  13
|          |          |          |          |                |
|          |          | SYMBOLIC | SYMBOLIC | SUBBLOCK TYPE__|  1 byte   14, 15
|          |          | SUBBLOCK | SUBBLOCK | DATA LENGTH    |
|          |          | AREA     |          | IN SUBBLOCK____|  3 bytes  16
|          |          |          |          | SUBBLOCK       |           17
|__________|__________|__________|__________| INFORMATION____|  (variable)
```

Figure A-5. The Symbolic Block, Version 0

Notes on Figure A-5:

1. The symbolic block contains all of the program's special debugging information.

2. The length of the symbolic block is fullword aligned. If there are no symbolic sections, the block is four bytes long.

3. The section area can contain any number of symbolic sections. Every code section can have a symbolic section. If it does, these sections are in the same order as the corresponding code sections.

4. The length reflects the length of data in the area. If the length is not divisible by four, up to three bytes of X'00' filler are 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.

5. The external name of the corresponding code section.

6. Doubleword-aligned object time length of the corresponding section block in the run block.

7. Location of the source file at compilation time.

8. Location of the listing file at compilation time (or blank).

9. The external reference pool 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 the Linker program without changing any locations in the symbolic section (other than adding or dropping the whole section).

10. Length of the external reference pool (in bytes), including this word.

11. External reference entry of which there may be any number.

12. External name in ASCII with trailing blanks.

13. Displacement within the code section to a 4-byte RCON.

14. The symbolic subblock area may contain any number of subblocks. Each subblock is composed of only one type of debugging information (i.e., statement number block).

15. The subblock types are language independent codes and are interpreted the same way for all languages.

16. The length of data in the subblock, which includes the first four bytes of the subblock. If this length is not divisible by four, up to three bytes of X'00' filler are 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.

17. 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, and contains language-independent information identifying individual statements in the program's section. One statement number subblock must be present in each symbolic section.

```
                                                               NOTES

|         | 1                                  |   1 byte        1
|         | LENGTH OF DATA IN DATA FIELD       |   3 bytes       2
|STATEMENT|            |          | LINE NUMBER|
|NUMBER   |ANY #       |STATEMENT |            |   2 bytes       3, 4
|BLOCK    |OF          |ENTRY     | CHARACTER  |
|         |STATEMENT   |          | STRING     |   5 bytes       5
|         |ENTRIES     |          | CODE SECTION|
|         |            |          | DISPLACEMENT|  3 bytes       6
```

Figure A-6. Statement Number Block, Version 0

Notes on Figure A-6:

1. 1 = statement number type subblock.

2. The length of data in the subblock including this word. If this length is not divisible by four, up to three bytes of X'00' filler are added to the end of the subblock to make the following subblock start on a fullword boundary.

3. Any number of statement entries may follow. 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 is one entry per verb in the COBOL source). The entries are in order of increasing displacements.

4. The line number in binary (no negative values) or zero to indicate the inline nonsymbolic code. The exact definition of this entry is language-dependent, but normally indicates the statement line number.

5. The character string (in ASCII with trailing blanks) performs 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 is used for an abbreviation of the COBOL verb.)

6. The run-time displacement into the section of the start of the statement.

Data Name Subblock

The data name subblock is generated by all high-level language translators to support symbolic access to data items at run time through command language facilities. One dataname subblock must be present in each symbolic section.

```
                                                                       NOTES
                                                              |
|     | 2 = DATA NAME SUBBLOCK                                |1 byte     1
|     | LENGTH OF DATA IN DATA FIELD                          |3 byte     2
|     |       | NUMBER OF INDEX ITEMS                         |4 bytes    3,4
|     | DATA  | ANY #  |                                      |
|     | NAME  | OF POS.| INDEX ITEM                           |
|     | INDEX | INDEX  |                                      |
|     |       | ITEMS  |                                      |4 bytes    5
|     |       |      |LENGTH OF ENTRY -1                      |1 byte     6,7
|DATA |       |      |      | INDEX OF EXTERNAL SECTION       |1 byte     8
|NAME |       |      | DATA | PATH TYPE OF DATA ITEM          |1 byte     9
|BLOCK|       |      | PATH | DISPLACEMENT TO THE DATA ITEM   |3 bytes    10
|     |       | DATA |      | VARI-| INDICATOR                |1 bit      11
|     |       | NAME | DATA | ABLE | INDICATOR                |1 bit      12
|     | ANY # | ENTRY| DES- | TYPE | FORMAT INDICATOR         |6 bit      13
|     |OF DATA|      | CRIP | SCALE                           |1 byte     14
|     | NAME  |      | TION | DATA ITEM LENGTH                |1 byte     15
|     |ENTRIES|      | DATA NAME LENGTH - 1                   |1 byte
|     |       |      | DATA NAME                              |variable   16
|     |       |      | OPTIONAL OFFSET                        |2 bytes    17
|     |       |      |      | NUMBER OF SUBSCRIPTS REQUIRED   |1 byte     18
|     |       |      |ARRAY | NUMBER OF DIMENSION DESCRIPTIONS|1 byte     19
|     |       |      |DESCR | DIMEN| HIGH BOUND OF SUBSCRIPT  |2 bytes    20
|     |       |      |IPTOR | SION |                          |
|     |       |      |      | DESCR| LENGTH OF SUBSCRIPT ITEM |2 bytes
```

Figure A-7. Data Name Subblock, Version 0

Notes on Figure A-7:

1. 2 = Data name type subblock.

2. Length of data in subblock (including this word). If this length is not divisible by four, up to three bytes of X'00' filler are added at the end of the subblock to make the next section start on a fullword boundary.

3. This index contains compiler-dependent information used to efficiently search the subblock for a given symbol.

4. Number of index items.

5. Index item = displacement within symbolic section to first dataname entry with the compiler-dependent indexed attribute.

6. Any number of data name entries. These can be in any order. It is expected that any one compiler orders these such that when the compiler is identified they can be efficiently searched.

7. Length of entry minus one.

8. 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. 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 referenced external section.

   - 2 = Displacement locates a 4-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).

10. Displacement from the indexed external section to the data item (for type 2 path, this is the displacement to the address constant).

11. Indicator -- referenced item (whether referenced using subscripts or not) is an elementary data item if equal to 1.

12. Indicator -- subscripts required if equal to 1.

13. 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 fields).
    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 decimal point character.
    9 = Binary COBOL halfword index value. (Length per occurrence number is in data item length; length is stored in 2 bytes: scale and data item length. Length stored is the length per occurrence number.)
    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.
    14 = Unsigned zone numeric.
    15 = BASIC array.
    16 = COBOL group item.
    17 = BASIC string scalar.
    18 = Binary COBOL fullword index value. (Length per occurrence number is in data item length. Length is stored in two bytes: scale and data item length. Length stored is the length per occurrence number.)
    19 = Logical (FORTRAN).
    20 = Complex (FORTRAN).

14. Scale -- A signed binary number indicating how far left of the rightmost digit the decimal point is to be relocated. Relocation is to the right for negative numbers. (For character type 1 fields, this byte is considered part of the item length.)

15. Data item length -- Specifies the length of the data time. If the data is character type, both this and the preceding byte are used for the length.

16. Data name -- If the data name must be qualified, all necessary levels of the name are listed with highest level first and the levels of qualifications separated by a point (.). If the name is qualified but is unique in the program, only the lowest level of the name should be listed.

17. Optional offset (present with data path 2 only).

18. Number of subscripts required. There is one dimension description for each subscript.

19. Indicated number of dimension descriptions.

20. Dimension description -- There is 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 is assumed to be 1.

A.2.3 The Linkage Block, Version 0

The linkage block 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 followed by any number of blocks of section information.

```
                                                                      NOTES

|        |LENGTH OF LINKAGE BLOCK (FULLWORD ALIGNED)            |  4 bytes 1,2
|        |        |        | LENGTH OF DATA IN THIS SECTION     |
|        |        | SECTION| BLOCK IN BYTES                     |  4 bytes 3,4,5
|        |        | BLOCK  | EXTERNAL NAME OF SECTION (ASCII)   |  8 bytes 6
|        |        |        | TYPE OF BLOCK 0=CODE 1=STATIC      |  1 byte  7
|        |        |        | OBJECT TIME LNGTH OF CORRESPONDING |
|        | CODE   |        | SECTION BLOCK IN THE RUN BLOCK     |
|        | SECTION|        | DOUBLEWORD ALIGNED                 |  3 bytes 8
|        | AREA   |        | COMPILER/ASSEMBLER NAME            |  2 bytes 9
|        |        |        | COMPILER/ASSEMBLER VERSION         |  3 bytes 10
|        |        |        | DATE OF COMPILATION OF THIS        |
|        |        |        | SECTION PACKED DECIMAL FORMAT      |  3 bytes 11
|        |        |        | LENGTH OF SYMBOLIC SECTION OR      |
|LINKAGE |        | LINKAGE| RUN TIME LENGTH OF STATIC AREA     |  4 bytes 12
| BLOCK  |        | SECTION| ENTRY| LENGTH OF ENTRY POINT LIST  |  4 bytes 13,14
|        |        | BLOCK  | POINT|ANY #    | ENTRY POINT NAME  |  8 bytes 15,16
|        |        | (PADDED| LIST |OF ENTRY | (ASCII)           |
|        |        |TO FULL-|      |POINT    | RUN-TIME OFFSET OF|  4 bytes 17
|        |        | WORD)  |      |ITEMS    |ENTRY PT IN SECTION|
|        |        |        |       | LENGTH OF RELOCATION       |  4 bytes 18
|        |        |        | RELOC-| REFERENCE BLOCK            |
|        | STATIC |        | ATION |       | LENGTH OF EXTERNAL |  4 bytes 19,20
|        | SECTION|        | REFER-| EXTER-| NAMES BLOCK        |
|        | AREA   |        | ENCE  | NAL   | ANY #              |  8 bytes 21,22
|        |        |        | BLOCK | NAMES | OF EX-   EXTERNAL  |
|        |        |        |       | BLOCK | TERNAL   NAME      |
|        |        |        |       |       | NAMES              |
|        |        |        |       | RELOC-| ANY #              |
|        |        |        |       | ATION | OF RE-  RELOCATION |
|        |        |        |       | ITEM  |LOCATION ITEM       |
|        |        |        |       | LIST  | ITEMS              |  5 bytes 23
```

Figure A-8. The Linkage Block, Version 0

Notes on Figure A-8:

1. The fullword aligned length of the linkage block.

2. The code section block area is composed of one section block for each code section in the program. The blocks are in the order in which the code sections appear in the code block of the run block.

3. The static section block area 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.

4. The code and static section blocks in the linkage area have sufficient information so sections can be added or deleted from a program, and so 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 is presented, and the differences are noted in the skeleton.

5. The length of data in the section block in bytes reflects the length of data in the area. If the length is not divisible by four, up to three bytes of X'00' filler are 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.

6. The external name of the section in ASCII with trailing blanks.

7. The type of block:

   - 0 = Code
   - 1 = Static

8. The object-time length of the corresponding section block in the run block. Doubleword aligned.

9. Compiler name or designation in ASCII with trailing blanks.

10. Version and modification level of the compiler in packed decimal.

11. Date of compilation of this section in packed decimal (YYDDD).

12. For a code section block, this is the length of the corresponding symbolic section (or 0 if there is no corresponding symbolic section), fullword-aligned.

    For a static section block, this is the run-time length of the program static area, doubleword-aligned.

13. The entry point list is a list of all names in this section that are known outside of the section. This list may have any number of entries.

14. The length of the entry point list including this word.

15. The list may contain any number of names ,and address pairs. They may be in any order, but must not be repeated.

16. The entry point name in ASCII with trailing blanks.

17. The run-time displacement into the section for this entry point.

18. The relocation reference block lists all locations within the section that 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.

19. 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.

20. Length of the external names block including this word.

21. List of external reference names. These names can be in any order, but they are referenced by their position in the list. The first name is number one.

22. External name in ASCII with trailing blanks.

23. List of relocation items. This list is in order of increasing displacement. There can be any number of entries in this list.

The 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 are listed in the relocation reference 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.

				NOTES
RELOCA-TION ITEM	FLAG BYTE	RESERVED; MUST BE ZERO	2 bits	
		RCON IF =1	1 bit	1
		ADDRESS IS RELOCATION RCD	1 bit	2
		LENGTH OF ADDRESS CONSTANT	1 bit	3
		DIRECTION OF RELOCATION	1 bit	4
		UNRESOLVED FLAG	1 bit	5
		RESERVED; MUST BE ZERO	1 bit	
	OBJECT TIME DISPLACEMENT INTO THE SECTION OF THE TARGET ADDRESS CONSTANT OR RELOCATION RECORD.		3 bytes	6
	NUMBER OF EXTERNAL NAME REFERENCED		1 byte	7

Figure A-9. Relocation Reference Block, Version 0

Notes on Figure A-9:

1. If the reference is an RCON, the referenced name must be in a static section. If the address constant is in a static section, the relocation record is the do-not-relocate bit set.

2. 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, have this bit set.

3. If the target is within a code section, this bit indicates the length of the target address constant:

   - 0 = Three bytes
   - 1 = Four bytes

   If the target is a relocation record, this bit is ignored.

4. Direction of relocation:

   - 0 = Positive (add the address of the start of the section to the specified location)
   - 1 = Negative (subtract from the location)

5. Unresolved flag. If set, the address is resolved relative to an address of X'F00000'.

6. Object time displacement into the section of the target address constant or relocation record.

7. Order 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).

A.3 OBJECT FILE FORMAT FOR RELEASE 7.00 SERIES

This section describes Version 1 of the object file format for the Release 7.00 series of the VS Operating System. Refer to Section A.2 for a description of Version 0.

A.3.1 The Run Block

The run block contains the information needed by the system to run a program. It is 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 when the program starts.

				BYTES	NOTES
RUN BLOCK	CODE AND PROLOG BLOCK	PROLOG BLOCK	SEE FIGURE A-11		
		CODE BLOCK	CODE SECTION	variable	1, 2
	LENGTHS BLOCK	SEE FIGURE A-12			
	STATIC BLOCK	See FIGURE A-13 Padded to fullword			
	MODULE BLOCK	SEE FIGURE A-14			

Figure A-10. The Run Block, Version 1

Notes on Figure A-10:

1. The code block can contain any number of code sections. This block contains all of the executable instructions in the program and can contain no modifiable data. The code block is composed of any number of sections, where a section is a contiguous area of code that can be moved as a whole by the Linker. There is no requirement for a particular order of the sections within the code block.

2. The code section is a block externally identified by its name. It is an independent contiguous area of code supplied by the language translator. The first location is on a doubleword boundary, and the length is divisible by eight. Address constants must be resolved prior to execution so that the program can be run without changing any locations in the section.

The Prolog Block

				BYTES	NOTES
64 Bytes	PROLOG BLOCK		LENGTH OF CODE AND PROLOG	3.5	1
			OBJECT FORMAT NUMBER	0.3	2
			PROGRAM ENTRY POINT ADDRESS	4.0	3
			PROGRAM BASE ADDRESS	4.0	4
		INTERPRETER INFORMATION	INTERPRETER FLAG	0.1	5
			RESERVED (0)	0.7	
			INTERPRETER DEBUG ENTRY ADDRESS	3.0	6
		PROGRAM INFORMATION	NAME (ASCII)	40.0	7
			VERSION # (packed dec)	4.0	8
			PATCH FLAG	0.1	9
			CODE RELOCATABILITY FLAG	0.1	10
			RESERVED (0)	0.6	
			RELEASE DATE *	3.0	11

* Packed decimal

Figure A-11. The Prolog Block, Version 1

Notes on Figure A-11:

1. The length of the code and prolog block is used to find the start of the static and lengths blocks. The length field of the combined code and prolog block is the number of bytes in the combined block.

2. In format Version 0, the value of the length was kept in bytes but required to be a multiple of eight indicating doubleword alignment. By placing the object version number adjacent to this length field and assigning the value of 0 to the original format, all older object code files do have an object version number and it is implicitly 0.

3. The entry point address is the address to which control is passed when the program is started. It is 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). The entry point address indicates the starting point for the execution of the program. A value of X'100040' (assuming a base address of X'100000') points to the first byte in the CODE block. This value is the default program entry point address.

4. The program base address follows the program starting address in the extended prolog block. It indicates the absolute address of the first byte of the program file.

5. The interpreter flag indicates whether (=1) or not (=0) the program is an interpreter.

6. This is the initialization entry address for the Debugger. It is valid only when the interpreter flag is set. The Debugger uses the initialization entry to set up for debugging the interpretive language source program.

7. This field is a 40-byte field for the program name. The default value of the field is a string of blanks.

8. The program version number in packed decimal. The default value of the field is 0 in packed decimal.

9. This field is a flag to indicate whether (=1) or not (=0) the program has been patched.

10. This field is a flag to indicate whether (=1) or not (=0) the code segment can be relocated, that is, it contains no absolute address references within the code itself. This is required of all code files in order to qualify for use as an SSL.

11. The program release date in packed decimal. The default values of the date field is the system date at the time of creation in packed decimal. The format is YYDDD.

The Lengths Block

		BYTES	NOTES
LENGTHS BLOCK	OBJECT TIME STATIC BLOCK LENGTH LESS PAD	4	1
	RUNTIME STATIC BLOCK LENGTH MULTIPLE OF EIGHT	4	2
	MODULE BLOCK LENGTH; FULLWORD LENGTH; 0 IF EMPTY	4	3
	RESERVED; ONE FULLWORD; MUST BE 0	4	

Figure A-12. The Lengths Block, Version 1

Notes on Figure A-12:

1. Length of the static block in bytes of the file reflects the length of data in the static area at object time. If the length is not divisible by four, up to three bytes of slack are 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..

2. Length of the runtime memory image of the static area when mapped.

3. Length of the module block which follows the static block. The module block can have a length of 0.

The Static Block

The static block contains initial value records that are to be processed by the program startup facility in the operating system. These records 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 are resolved by Linker or Translator programs as if they were in a code section. Address constants that address locations within the static sections are resolved as if the start of the static block were location 0.

				Bytes	NOTES
STATIC BLOCK	STATIC SECTION	INITIAL VALUE RECORD	LENGTH OF DATA IN DATA FIELD	1	1, 2, 3, 4
			RECORD TYPE	4	5
			RUN TIME DISPLACEMENT	2	6
			DATA FIELD SEE FIGURE A-14	1-256	7

Figure A-13. The Static Block, Version 1

Notes on Figure A-13:

1. The static block may contain any number of static sections, including zero. The static block contains sections of initial value records.

2. 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 have initial values, there are no records for that section and the object time length is zero. The length of these static sections does not correspond to the length of the expanded static section at runtime. In order to distinguish between the object time locations and runtime locations, the following naming convention is used: locations in the object code file are referred to as object time locations or are specified by their object time address; locations that are used during running of a program are referred to as runtime locations or are specified by their runtime addresses. Because the code block is without change at runtime, this distinction is normally not made for any locations in the code block. When descriptions apply to both static and code areas, runtime and object time may be used interchangeably to refer to the code area.

3. Initial value records specify locations that are to have initial values in the named static section. The initial value records also specify the values the program startup mechanism is to assign to these locations. There are five types of initial value records:

   - The origin record, which specifies how far from the start of the expanded run time static area this section starts

   - The value record, which specifies the value to be placed in the static area

- The relocation record, which specifies that program startup is to supply the address of a run time location in a static area

- 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

- The compressed record, which specifies the compressed value to be expanded and placed in the static area

4. Length of data within the data field in this record minus one. This field is not used for the compressed record type.

5. The record type:

   - 0 = Value
   - 1 = Origin
   - 2 = Relocation
   - 4 = Repeated
   - 8 = Compressed

6. Runtime displacement. This field can be interpreted in two ways. For origin records (record type 1), this field indicates the displacement from the start of the static area of this static section. For all other record types, this field indicates the runtime displacement from the start of the static section of the record's data.

7. Data field. The length and format of this field vary depending on the record type. Refer to Figure A-14 for the different data fields.

The data field for:

Value Record:

		BYTES	NOTES
DATA FIELD	DATA TO BE MOVED TO THE STATIC SECTION	1 - 256	1

Origin Record:

		BYTES	NOTES
DATA FIELD	DUMMY DATA	1	

Relocation Record:

		BYTES	NOTES
DATA FIELD	3	1.0	
	B'0010'	0.4	
	RUNTIME DISPLACEMENT OF DATA WITHIN THE STATIC SECTION	2.4	
	RESERVED (0)	0.1	
	RELOCATE CODE COUNT	0.2	2
	CODE DIRECTION FLAG	0.1	
	LENGTH FLAG	0.1	3
	STATIC DIRECTION FLAG	0.1	4
	RELOCATE STATIC FLAG	0.1	5
	INITIAL VALUE	3.1	6

Repeated Record:

		BYTES	NOTES
DATA FIELD	REPETITION FACTOR	2	7
	DATA TO BE REPEATED WITHIN THE STATIC AREA	1 - 256	8

Compressed Record:

		BYTES	NOTES
DATA FIELD	LENGTH OF COMPRESSED DATA	2	9
	COMPRESSED DATA	1 - 2048	10

Figure A-14. The Data Field, Version 1

Notes on Figure A-14:

1. This data is moved unchanged to the runtime static area.

2. The code relocation count. This count indicates whether or not the associated initial value is based on the address of the start of the code area. If a nonzero count is indicated (=1, =2, or =3), the static item must be relocated with respect to any change to the program base address at either link or execution time. If the count is reset (=0), there is no code relocation required. A code direction flag has been added to indicate whether the code relocation is to be applied as a positive (=0) or negative (=1) factor.

3. Length of target address constant:

   - 0 = three bytes
   - 1 = four bytes

4. Direction of relocation (applies to both code and static direction flags):

   - 0 = positive
   - 1 = negative

5. The do-not-relocate flag is used if the address is unresolved, if the address referenced a code location, or if this is an R type address constant.

   - 0 = Relocate the address constant.
   - 1 = Move the address constant to the specified location but do not relocate with respect to the static area.

6. Initial value of the address constant. If the target is three bytes long, only the last three bytes are moved to the target area and the high-order bit is ignored. If the target is four bytes long, the high-order bit is propagated through the seven remaining bits of the high-order target byte.

7. The repetition factor may have a value from 2 through 32767.

8. The data to be repeated within the static area.

9. The length of the compressed data may have a value from 1 through 2048.

10. The compressed data is expanded before being moved to the static area.

The Module Block

The module block is a new block designed to support the SSL feature and the subroutine library search by entry name feature of VS Operating System Release 7 (See Figure A-15). The block holds three tables. The first table, the Entry Point Reference table, contains an ordered list of entries, each being the information about an entry point in the program. The second table, the Global External Name table, holds all the external names referenced from a static section which are as yet unresolved in the program and also have an alias assigned. The third table, the Global Reference Index table, contains the locations where these static references to the external names are made.

```
                                                               BYTES      NOTES

 _________________________________________________________
|          |          | EPR TABLE SIZE (4 IF EMPTY)       |     4.0          1
|          |          |ANY # OF | ENTRY POINT NAME        |
|          |ENTRY     |LEXICALLY| (ASCII)                 |    40.0          2
|          |POINT     |ORDERED  |1* |  3 bits | 4 bits ** |
|          |REFERENCE |ENTRY PT |   | Resrvd. |           |     1.0          3
|          |TABLE     |REFERENCE| RUN TIME OFFSET OF ENTRY|
|  MODULE  |          |ENTRIES  | POINT/PGM BASE ADDRESS  |     3.0          4
|  BLOCK   |          | GEN TABLE SIZE (4 IF EMPTY)       |     4.0          5
|          |GLOBAL    |ANY # OF     | EXTERNAL NAME (ASCII)|    40.0          6
|          |EXTERNAL  |LEXICALLY    | ALIAS               |
|          |NAME      |ORDERED      |                     |    40.0          7
|          |TABLE     |GLBL EXTERN. | INITIAL GRI INDEX   |
|          |          |NAME ENTRIES |                     |     4.0          8
|          |          | GRI TABLE SIZE (4 IF EMPTY)       |     4.0          9
|          |GLOBAL    | ANY # OF  | RUN TIME OFFSET OF    |
|          |REFERNCE  | GLOBAL    | EXTERNAL REFERENCE IN |
|          |ITEMS     | REFERENCE | STATIC AREA           |     4.0         10
|          |TABLE     | ITEMS     | NEXT GRI INDEX        |
|__________|__________|___________|_(OR END OF LIST)______|     4.0         11
```

* 1 bit -- initialization intercept
0 = indicates not an intercept;
1 = indicates entry point is an initialization intercept.

** Relative process level (default value = 0)

Figure A-15. The Module Block, Version 1

Notes on Figure A-15:

1. Entry Point Reference table (EPR): When a program file is utilized as an SSL, the EPR table is used by the operating system to resolve references to this SSL from a user program. When an SSL file is created, any subset of the program entry points may be designated for inclusion in the EPR. A second use of the EPR is to determine whether a program file contains an entry point which will resolve an, as yet unresolved, external reference during Linker processing. This is the Subroutine Library search process. An EPR table entry consists of three fields: Entry Point Name, Flags Byte and Entry point offsets. The entries are lexically ordered according to the entry point names. If the EPR table has no entries, the length of the EPR table is 4 bytes.

2. A 40-byte field that contains the entry point name in ASCII.

3. The second field is one byte long. The first half of the byte is a set of flags. The most significant bit is an initialization intercept flag. The remaining bits are unassigned and set to 0. When set (=1), the initialization intercept flag indicates to the operating system that at the time of mapping the SSL, the operating system executes the code at the entry point to initialize the SSL. The second half of the byte holds the relative process level of the entry point for the use of the operating system when creating the SSL.

4. The offset of the entry point relative to the program base address. The field is 3 bytes long.

5. Global External Name table (GEN): The purpose of this table is to enable the operating system to determine which shared subroutines are referenced by a program. If the program does not have any SSL references (unresolved locally within the program and no alias assigned to the external reference name), then the GEN table will be empty. The table can hold any number of entries. The entries are lexically ordered according to the names in the entries. Each entry consists of three subfields: External name, Alias and GRI Index.

6. External name referenced by the program. The name field is 40 bytes long.

7. The alias name that is used to determine the correct SSL to resolve all references to the external name. The alias field has a length of forty bytes.

8. The initial GRI index that is used to find the first member of the GRI table for this external name. This field is 4 bytes long. The value is an 'index' rather than an 'offset'.

9. GRI table: The GRI table is designed to enable the operating system to resolve all references to an external name after the SSL associated with the name has been located and mapped. The first item of the table is a 4-byte length field. When the table is empty, the value in the length field is 4.

10. Global reference items (GRI). An item uniquely identifies the position (runtime offset) in the static section where an external reference is made. For every entry in the GEN table, there is at least one GRI table entry. Since an external name can be referenced more than once, there may be more than one GRI entry associated with a GEN entry. The GEN entry points to the first GRI entry which is the first of the linked list of other GRI entries for the same name.

11 Next GRI index used for chaining the list. For every GEN entry, there is a linked list of GRI entries.

A.3.2 The Symbolic Block, Version 1

The symbolic block is a pool of information used by the system's debugging program. The block is partitioned into a length field and any number of section-related blocks.

```
                                                                  BYTES  NOTES

 _____________________________________________________________
|         | LENGTH OF SYMBOLIC BLOCK                           |
|         | FULLWORD ALIGNED__________________________________ |   4     1, 2
|         |         | LENGTH IN BYTES OF                      |
|         |         | DATA IN THIS SECTION___________________ |   4     3, 4
|         |         | EXTERNAL NAME OF                        |
|         |         | CORRESPONDING CODE SECTION_____________ |   8     5
|         |         | OBJ TIME LENGTH OF                      |
|         |         | SECTION IN RUN BLOCK                    |
|         |         | DOUBLEWORD ALIGNED_____________________ |   4     6
|         |         |         |                               |
|         |         | SOURCE  | FILE NAME__________________   |   8     7
|         |         | FILE    |                               |
|         |         | LOCATION| LIBRARY NAME_______________   |   8
|         |         |         |                               |
|         |         |_________| VOLUME NAME________________   |   6
|         |         |         |                               |
|         | SYMBOLIC| LISTING | FILE NAME__________________   |   8     8
| SYMBOLIC| SECTION | FILE    |                               |
| BLOCK   | AREA    | LOCATION| LIBRARY NAME_______________   |   8
|         |         |         |                               |
|         |         |_________| VOLUME NAME________________   |   6
|         |         |         |                               |
|         |         |EXTERNAL | LENGTH OF POOL_____________   |   4     9, 10
|         |         |REFERENCE| ANY # OF  | EXTERNAL          |
|         |         |POOL     | EXTERNAL  | NAME_____________ |   8     11, 12
|         |         |         | NAME      | CODE SEC DIS-     |
|         |         |_________| ENTRIES   | PLACEMENT________ |   4     13
|         |         |         |           |                   |
|         |         | SYMBOLIC| SYMBOLIC  | SUBBLOCK TYPE____ |   1     14, 15
|         |         | SUBBLOCK| SUBBLOCK  | DATA LENGTH       |
|         |         | AREA    |           | IN SUBBLOCK______ |   3     16
|         |         |         |           | SUBBLOCK          |         17
|_________|_________|_________|___________| INFORMATION______ | variable
```

Figure A-16. The Symbolic Block, Version 1

Notes on Figure A-16:

1. The symbolic block contains all of the program's special debugging information.

2. The length of the symbolic block is fullword aligned. If there are no symbolic sections, the block is four bytes long.

3. The section area can contain any number of symbolic sections. Every code section can have a symbolic section. If the code section does have a symbolic section, the symbolic sections are in the same order as the corresponding code sections.

4. The length reflects the length of data in the area. If the length is not divisible by four, up to three bytes of X'00' filler are 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.

5. The external name of the corresponding code section.

6. Doubleword-aligned object time length of the corresponding section block in the run block.

7. Location of the source file at compilation time.

8. Location of the listing file at compilation time (or blank).

9. The external reference pool 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 to reference that label. This structure allows modules to be added or dropped by the Linker program without changing any locations in the symbolic section (other than adding or dropping the whole section).

10. Length of the external reference pool (in bytes), including this word.

11. External reference entry of which there may be any number.

12. External name in ASCII with trailing blanks.

13. Displacement within the code section to a 4-byte RCON.

14. The symbolic subblock area may contain any number of subblocks. Each subblock is composed of only one type of debugging information (i.e., statement number block).

15. The subblock types are language-independent codes and are interpreted the same way for all languages.

16. The length of data in the subblock, which includes the first four bytes of the subblock. If this length is not divisible by four, up to three bytes of X'00' filler are 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.

17. 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, and contains language-independent information identifying individual statements in the program's section. One statement number subblock must be present in each symbolic section.

				BYTES	NOTES
STATEMENT NUMBER BLOCK	1			1	1
	LENGTH OF DATA IN DATA FIELD			3	2
	ANY # OF STATEMENT ENTRIES	STATEMENT ENTRY	LINE NUMBER	2	3, 4
			CHARACTER STRING	5	5
			CODE SECTION DISPLACEMENT	3	6

Figure A-17. Statement Number Block, Version 1

Notes on Figure A-17:

1. 1 = Statement number type subblock.

2. The length of data in the subblock including this word. If this length is not divisible by four, up to three bytes of X'00' filler are added to the end of the subblock to make the following subblock start on a fullword boundary.

3. Any number of statement entries may follow. 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 is one entry per verb in the COBOL source). The entries are in order of increasing displacements.

4. The line number in binary (no negative values) or zero to indicate the inline nonsymbolic code. The exact definition of this entry is language-dependent, but normally indicates the statement line number.

5. The character string (in ASCII with trailing blanks) performs 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 is used for an abbreviation of the COBOL verb.)

6. The runtime displacement into the section of the start of the statement.

Data Name Subblock

The data name subblock is generated by all high-level language translators to support symbolic access to data items at runtime through command language facilities. One dataname subblock must be present in each symbolic section.

```
                                                                BYTES   NOTES

|     | 2 = DATA NAME SUBBLOCK                                |   1       1
|     | LENGTH OF DATA IN DATA FIELD                          |   3       2
|     |        | NUMBER OF INDEX ITEMS                        |   4       3,4
|     | DATA   | ANY #    |                                   |
|     | NAME   | OF POS.  | INDEX ITEM                        |
|     | INDEX  | INDEX    |                                   |
|     |        | ITEMS    |                                   |   4       5
|     |        |       |LENGTH OF ENTRY -1                    |   1       6,7
|DATA |        |       |       | INDEX OF EXTERNAL SECTION    |   1       8
|NAME |        |       | DATA  | PATH TYPE OF DATA ITEM       |   1       9
|BLOCK|        |       | PATH  | DISPLACEMENT TO THE DATA ITEM|   3      10
|     |        | DATA  |       |VARI-| REFERENCE INDICATOR    |  .1      11
|     |        | NAME  | DATA  |ABLE | ARRAY FLAG INDICATOR   |  .1      12
|     | ANY #  | ENTRY | DES-  |TYPE | FORMAT INDICATOR       |  .6      13
|     |OF DATA |       | CRIP  |SCALE                         |   1      14
|     | NAME   |       | TION  |DATA ITEM LENGTH              |   1      15
|     |ENTRIES |       | DATA NAME LENGTH - 1                 |   1
|     |        |       | DATA NAME (Spelling)                 | variable 16
|     |        |       |                                      |
|     |        |       | OPTIONAL ADDITIONAL INFORMATION      |
|     |        |       | (SEE FIGURE A-19)                    |
```

Figure A-18. Data Name Subblock, Version 1

Notes on Figure A-18:

1. 2 = Data name type subblock.

2. Length of data in subblock (including this word). If this length is not divisible by four, up to three bytes of X'00' filler are added at the end of the subblock to make the next section start on a fullword boundary.

3. This index contains compiler-dependent information used to efficiently search the subblock for a given symbol.

4. Number of index items.

5. Index item = displacement within symbolic section to first dataname entry with the compiler-dependent indexed attribute.

6. Any number of data name entries. These can be in any order. It is expected that any one compiler orders these such that when the compiler is identified they can be efficiently searched.

7. Length of entry minus one.

8. 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. 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 referenced external section.

   - 2 = Displacement locates a 4-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).

   - 4 = Immediate halfword integer value in two low-order bytes of runtime displacement field. These symbols are procedure or function names and the value indicates the line number in the listing where the procedure or function starts. Additional information follows the spelling fields (see below).

   - 5 = Automatic/dynamic variable.

   - 6 = CLE standard parameter.

   - 7 = PL/1-based variable.

   - 8 = Immediate parameter ('C')

   - 9 = Register variable - runtime displacement field contains the register number.

   - 10 = C-based variable (using '*' prefixes)

10. Displacement from the indexed external section to the data item (for type 2 path, this is the displacement to the address constant).

11. Indicator -- Referenced item (whether referenced using subscripts or not) is an elementary data item if equal to 1.

12. Indicator -- Subscripts required if equal to 1.

13. 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 fields).

    - 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 decimal point character.

    - 9 = Binary COBOL halfword index value. (Length per occurrence number is in data item length; length is stored in 2 bytes: scale and data item length. Length stored is the length per occurrence number.)

    - 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.

    - 14 = Unsigned zone numeric.

    - 15 = BASIC array.

    - 16 = COBOL group item.

    - 17 = BASIC string scalar.

- 18 = Binary COBOL fullword index value. (Length per occurrence number is in data item length. Length is stored in two bytes: scale and data item length. Length stored is the length per occurrence number.)

- 19 = Logical (FORTRAN).

- 20 = Complex (FORTRAN).

- 21 = Floating decimal

- 22 = Pointer

- 23 = Unsigned binary integer

14. Scale -- A signed binary number that indicates how far left of the rightmost digit the decimal point is to be relocated. Relocation is to the right for negative numbers. (For character type 1 fields, this byte is considered part of the item length.)

15. Data item length -- Specifies the length of the data time. If the data is character type, both this and the preceding byte are used for the length.

16. Data name -- If the data name must be qualified, all necessary levels of the name are listed with highest level first and the levels of qualifications separated by a point (.). If the name is qualified but is unique in the program, only the lowest level of the name should be listed.

Option Additional Information for:

Path type = 2:

		BYTES	NOTES
ADDITIONAL INFO	OPTIONAL OFFSET	2	1

Path type - 4:

		BYTES	NOTES
ADDITIONAL INFORMATION	END	2	2
	LEVEL	1	3
	SON	3	4
	BROTHER	3	5

Path type = 6:

		BYTES	NOTES
ADDITIONAL INFO	OFFSET	2	1

All other path cases when array flag indicator is on:

				BYTES	NOTES
ADDITIONAL INFORMATION	OPTIONAL OFFSET			2	1
	ARRAY DESCRIPTOR	NUMBER OF SUBSCRIPTS REQUIRED		1	6
		NUMBER OF DIMENSION DESCRIPTIONS		1	7
		DIMENSION DESCR	HIGH BOUND OF SUBSCRIPT	2	8
			LENGTH OF SUBSCRIPT ITEM	2	

Figure A-19. Optional Information, Version 1

Notes on Figure A-19:

1. Optional offset.

2. The integer value that indicates the line number of last statement in procedure or function.

3. The binary value that indicates the nesting level of this procedure (1=outermost)

4. Indicates offset from start of symbolic section to dataname entry for first procedure internal to this procedure; 0, if none.

5. Indicates offset from start of symbolic section to dataname entry for next procedure at the same level as this; 0, if none.

6. Number of subscripts required. There is one dimension description for each subscript.

7. Indicates number of dimension descriptions.

8. Dimension description -- There is 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 is assumed to be 1 (For the C programming language, assumed to be zero).

A.3.3 The Linkage Block, Version 1

The linkage block 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 followed by any number of blocks of section information.

```
                                                                        BYTES  NOTES
 ______________________________________________________________________
|         |LENGTH OF LINKAGE BLOCK (FULLWORD ALIGNED)                  |  4.0   1,2
|         |         |LINKAGE SECTION BLOCK LENGTH (LESS PAD)           |  4.0   3,4,5
|         |         |        |CODE/STATIC SECTION NAME (ASCII)         | 40.0   6
|         |         |        |SECTION TYPE                             |  1.0   7
|         |         |        | OBJECT TIME LENGTH                      |
|         |         |        | IN RUN BLOCK                            |  3.0   8
|         |         |        | COMPILA-| COMPILER NAME                 | 12.0   9
|         |         |        | TION IN-| COMPILER VERSION (**)         |  4.0   10
|         |         |        | FORMA - | COMPILATION TIME (*)          |  4.0   11
|         |         |        | TION    | COMPILATION DATE (*)          |  4.0   12
|         |         |        |                                         |
|         |ALL      |        | SYMBOLIC SECTION FULLWORD /RUN-         |
|         |LINKAGE  |        | TIME STATIC SECTION                     |  4.0   13
|         |SECTIONS |        | LENGTH                                  |
| LINKAGE |ARE IN   |LINKAGE |                                         |
|  BLOCK  |THE SAME |SECTION | ENTRY | LENGTH OF ENTRY POINT LIST      |  4.0   14, 15
|         |ORDER AS |BLOCK   | POINT |ANY #     | ENTRY POINT NAME     |
|         |THE COR- |(PADDED | LIST  |OF ENTRY  | (ASCII)              | 40.0   16, 17
|         |RESPOND- |TO FULL-|       |POINT     | RUN-TIME OFFSET OF   |
|         |ING CODE |WORD)   |       |ITEMS     |ENTRY PT IN SECTION   |  4.0   18
|         |STATIC   |        |        | LENGTH OF RELOCATION           |
|         |SECTIONS |        | RELOC- | REFERENCE BLOCK                |  4.0   19
|         |IN THE   |        | ATION  |       | LENGTH OF EXTERNAL     |
|         |RUN      |        | REFER- | EXTER-| NAMES BLOCK            |  4.0   20, 21
|         |BLOCK    |        | ENCE   | NAL   | ANY #    |             |
|         |         |        | BLOCK  | NAMES | OF EX-   | EXTERNAL    |
|         |         |        |        | BLOCK | TERNAL   | NAME        |
|         |         |        |        |       | NAMES    |             | 40.0   22, 23
|         |         |        |        | RELOC-| ANY #    |             |
|         |         |        |        | ATION | OF RE-   |RELOCATION   |
|         |         |        |        | ITEM  |LOCATION  |  ITEM       |
|_________|_________|________|________| LIST  | ITEMS    |_____________|  5.0   24
```

*   Packed decimal
**   Unsigned packed

Figure A-20. The Linkage Block, Version 1

Notes on Figure A-20:

1. The fullword aligned length of the linkage block.

2. The code section block area is composed of one section block for each code section in the program. The blocks are in the order in which the code sections appear in the code block of the run block.

3. The static section block area 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.

4. The code and static section blocks in the linkage area have sufficient information so sections can be added or deleted from a program, and so 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 is presented, and the differences are noted in the skeleton.

5. The length of data in the section block in bytes reflects the length of data in the area. If the length is not divisible by four, up to three bytes of X'00' filler are 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.

6. The external name of the section in ASCII with trailing blanks.

7. The type of block:

   - 0 = Code
   - 1 = Static

8. The object-time length of the corresponding section block in the run block. Doubleword aligned.

9. Compiler name or designation in ASCII with trailing blanks.

10. Version and modification level of the compiler in packed decimal.

11. A 4-byte field which holds the time of compilation in packed decimal format.

12. Date of compilation of this section in packed decimal (YYDDD).

13. For a code section block, this is the length of the corresponding symbolic section (or 0 if there is no corresponding symbolic section), fullword-aligned.

    For a static section block, this is the runtime length of the program static area, doubleword-aligned.

14. The entry point list is a list of all names in this section that are known outside of the section. This list may have any number of entries.

15. The length of the entry point list including this word.

16. The list may contain any number of names and address pairs. They may be in any order, but must not be repeated.

17. The entry point name in ASCII with trailing blanks.

18. The runtime displacement into the section for this entry point.

19. The relocation reference block lists all locations within the section that 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.

20. 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.

21. Length of the external names block including this word.

22. List of external reference names. These names can be in any order, but they are referenced by their position in the list. The first name is number one.

23. External name in ASCII with trailing blanks.

24. List of relocation items. This list is in order of increasing displacement. There can be any number of entries in this list.

The 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, are listed in the relocation reference 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.

				NOTES
RELOCATION ITEM	FLAG BYTE	RESERVED; MUST BE ZERO	2 bits	
		RCON IF =1	1 bit	1
		ADDRESS IS RELOCATION RCD	1 bit	2
		LENGTH OF ADDRESS CONSTANT	1 bit	3
		DIRECTION OF RELOCATION	1 bit	4
		UNRESOLVED FLAG	1 bit	5
		RESERVED; MUST BE ZERO	1 bit	
	OBJECT TIME DISPLACEMENT INTO THE SECTION OF THE TARGET ADDRESS CONSTANT OR RELOCATION RECORD.		3 bytes	6
	NUMBER OF EXTERNAL NAME REFERENCED		1 byte	7

Figure A-21. Relocation Reference Block

Notes on Figure A-21:

1. If the reference is an RCON, the referenced name must be in a static section. If the address constant is in a static section, the relocation record has the do-not-relocate bit set.

2. 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, have this bit set.

3. If the target is within a code section, this bit indicates the length of the target address constant:

   0 = three bytes
   1 = four bytes

   If the target is a relocation record, this bit is ignored.

4. Direction of relocation:

   - 0 = Positive (add the address of the start of the section to the specified location)

   - 1 = Negative (subtract from the location)

5. Unresolved flag. If set, the address is resolved relative to an address of X'F00000'.

6. Object time displacement into the section of the target address constant or relocation record.

7. Order number of referenced external name. This number is either 0 (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 1).

A.4 TRANSLATOR PROCESSING

The object program is produced by a language translator. At this time, all address constants are resolved, or marked as unresolved. The program can then optionally be processed by the Linker program. The Linker adds, rearranges or deletes 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 can be run 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 generates the symbolic section of information for debugging.

- The run block -- Contains all of the information required to run the program. The operating system uses this part of the file for the code section 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.

- The symbolic block -- Contains information to aid in runtime debugging of the program. This area is used by the debugging facilities in the Help processor.

- The linkage block -- Contains information used by the Linker program. After adding or deleting sections, the Linker program relocates the address constants in the program using this information.

A.5 LINKER PROCESSING

A program file may be viewed as an ordered collection of named program sections. When the Linker operates on program files, it merely rectifies all address references of all input program sections and combines them into a new program file. Exceptions are that only one section of any unique section name may appear in the output file. The selection rule may be by input order or other selection criteria, depending upon the section types involved. Replacement is usually accomplished by preceding the program input with those sections that the user wants to replace. Explicit deletion of a section is also provided.

The order of the output sections is (1) all included code sections in the order in which they were input, followed by (2) all static (including named and blank common) sections in the order in which they were input. The order is reflected in each of the three resultant program blocks (run, symbolic, and linkage).

When the Linker program processes an object program, it performs the following operations to the basic parts of the object program:

- The run block -- Locations within the sections are not 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 replaced from a program, all relocation records that reference external symbols no longer present in that section must be adjusted. If a section is added, all references to it must be adjusted.

- The linkage block -- When a code or static section is added or replaced, the Linker adds or replaces the corresponding linkage section. It also adjusts the object time and runtime 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.

- The symbolic block -- If a section is added or replaced the Linker adds or replaces the corresponding symbolic section. It does not inspect or change any of the records within a section for any reason. The user has the option to exclude all symbolic sections.

A.6 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 the code section. The first location is location 1,048,576 (hexadecimal 100000). The system then doubleword-aligns the stack, determines the runtime length of the static area, and pushes this much space onto the stack. The start of this area is passed to the program in register 14. The initial value records are then processed. When an origin record is encountered, the origin displacement is added to the value in register 14 and this is used as the starting location of the section. When either text or relocation records are read, their displacement is 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 B
GLOSSARY

Address constant
A value, or an expression that represents a value, used in the calculation of storage addresses.

Address translation
Translation of a virtual address to its corresponding physical address.

Alias
A name of up to 40 characters that is assigned to each shared subroutine library. The system administrator does the assignment.

Alignment
The storing of data in relation to certain machine-dependent boundaries.

Alphanumeric
Pertaining to a character set that contains letters, digits, and usually other characters, such as punctuation marks.

American National Standards Institute (ANSI)
An organization created to establish voluntary industry standards.

Assistance ID (AID)
A special control character generated by a system communication key (HELP, PF key, or ENTER), which identifies the key used to initiate a program interrupt.

Background processing
A time-independent, noninteractive processing environment that consists of one or more programs and procedures run by a controlling procedure.

Block
A number of physical records, each 2048 (2k) bytes in length, which make up a disk volume.

Buffer
A memory area that temporarily holds blocks of data retrieved from disk.

Call

The action of bringing a computer program, a routine, or a subroutine into effect, usually be specifying the entry conditions and jumping to an entry point.

Child task

A task that has been created by another task through the TINVOKE system service. The child task must be completed before the parent task can be terminated.

Clock

A device that measures and indicates time. A register value that changes at regular intervals in such a way as to measure time.

Clock comparator

A hardware feature that causes an interruption when the time-of-day clock has equaled or exceeded the value specified by a program or virtual machine.

Contiguous

Touching or joining at the edge or boundary; adjacent. For example, an unbroken consecutive series of storage locations.

Control blocks

A series of tables used by the operating system for data storage.

Control section (CSECT)

That part of a program specified by the programmer to be a relocatable unit, all elements of which are to be loaded into adjoining main storage locations.

Currency symbol

A graphic character used to designate monetary quantities, for example $.

Cylinder

In a disk pack, the set of all tracks with the same nominal distance from the axis about which the disk pack rotates. The tracks of a disk storage device that can be accessed without repositioning the access mechanism.

Data Management System (DMS)

The VS file management system that allows developers to create, read, update and copy data files on a variety of storage media.

Deadlock

Unresolved contention for the use of a resource. An error condition in which processing cannot continue because each of two elements of the process is waiting for an action by or a response from the other.

Default

An alternative value, attribute, or option that is assumed when none has been specified.

Delimiter

A flag that separates and organizes items of data. Synonymous with punctuation symbol or separator. A character that groups or separates words or values in a line of input.

Diagnostic

Pertaining to the detection and isolation of a malfunction or mistake.

Diagnostic program

A computer program that recognizes, locates and explains either a fault in equipment or a mistake in a computer program.

Direct address

An address that designates the storage location of an item of data to be treated as an operand. Synonymous with one-level address.

Dual density

A feature that allows a program to use a tape unit in either 800- or 1600-byte-per-inch recording.

Dummy control section (DSECT)

A control section that an assembler can use to format an area of storage without producing any object code.

Dump

To write the contents of storage, or of part of storage, usually from an internal storage to an external medium, for a specific purpose (such as to allow other use of the storage as a safeguard against faults or errors, or in connection with debugging).

Dynamic allocation

Assignment of system resources to a program at the time the program is executed rather than at the time it is loaded into main storage. An allocation technique in which the resources assigned for the execution of computer programs are determined by criteria applied at the moment of need.

Event

An occurrence of significance to a task; typically, the completion of synchronous operation, such as an input/output operation.

Exception

An abnormal condition such as an I/O error encountered in processing a data set or a file.

Executable program

A program that has been link-edited and can therefore be run in a processor.

Expression

In assembler programming, one or more operations represented by a combination of terms and paired parentheses. A notation, within a program, that represents a value: a constant or a reference appearing alone, or combinations of constants and references with operators.

File

A set of related records treated as a unit.

File access mode

A mode that determines whether the file can be used as read-only or read/write.

File attribute

Any of the attributes that describe the characteristics of a file.

File descriptor record (FDR)

A file system data structure located within the VTOC that describes the attributes of a file. There is at least one FDR (format 1) for each file on a volulme.

Foreground job

A high-priority job, usually a real-time job. An interactive or graphic display job that has an indefinite running time during which communication is established with one or more users at local or remote terminals.

General register

A register used for operations such as binary addition, subtraction, multiplication and division. General registers are used primarily to compute and modify addresses in a program.

Global

Pertaining to that which is defined in one subdivision of a computer program, and used in at least one other subdivision of that computer program.

Halfword

A contiguous sequence of bits or characters that comprise half a computer word and can be addressed as a unit.

Halfword boundary

Any storage position address which, in decimal form, is evenly divisible by 2.

Immediate instruction

An instruction that contains within itself an operand for the operation specified, rather than an address of the operand.

Indirect address

An address that designates the storage location of an item of data to be treated as the address of an operand, but not necessarily as its direct address.

Intertask message system (ITM)

A group of supervisor calls that enable one task to dynamically exchange data with another task.

Invoke

To activate a program or procedure at one of its entry points.

I/O status word (IOSW)

A system control word located in memory location X'00' which reflects the result of an I/O operation. From one to eight bytes of information may be stored, depending upon the device type.

I/O processor (IOP)

The processor that controls the transfer of data between devices and main memory.

JSCI instruction

An assembly language instruction, jump to subroutine on condition indirect. A specified condition must be met before this instruction is executed.

JSI instruction

An assembly language instruction, jump to subroutine on any condition. This instruction is similar to the JSCI instruction, except that the jump to the subroutine is always made.

Link level

The environment associated with an invocation of the LINK system service.

Macroinstruction

A user-defined assembler instruction that expands into one or more machine instructions.

Modifiable data area

The area that each user is assigned to store variable data and dynamically initialized variables. This area is divided into the program stack and an I/O buffer area. The default modifiable data area size is set with the GENEDIT utility, but can be overridden with the SECURITY utility.

Module

A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to, or output from, an assembler, compiler, linkage editor, or executive routine.

Object code

Output from a compiler or assembler which is itself executable machine code or which can be processed to produce executable machine code.

Object module

The machine language code that results from a program compilation.

Op code

The mnemonic representation of an assembler language instruction.

Parent task

A task which invokes a subtask through issuing the TINVOKE system service.

Port

An access point for data entry or exit. That part of a processor that is dedicated to a single data channel for the purpose of receiving data from or transferring data to one or more external or remote devices.

Process

A unique, finite course of events defined by its purpose or by its effect, achieved under given conditions. An executing function, or a function that is waiting to be executed.

Process level

A PCW field which determines the level of privilege for executing code. The process level determines the access rights to certain areas of the virtual address space.

Program

A set of actions or instructions that a machine can interpret and execute.

Ready list

A chain of elements that represents the work to be performed.

Ready state

A state in which a task is ready to be activated and is contending for processor execution time.

Recursive routine

A routine that may be used as a routine of itself, calling itself directly or being called by another routine. The use of a recursive routine usually requires that records are kept for example, in a pushdown list) on the status of its unfinished uses.

Reentrant

The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

Region

A contiguous portion of a task's virtual address space that begins on a page boundary and contains a variable number of pages. Although the number of regions can vary, a system's virtual address space is broken up into a maximum of 64 regions.

Relocatable

The attribute of a set of code whose address constants can be modified to compensate for a change in origin.

Return code

A code used to influence the execution of succeeding instructions.

Ring memory protection

A VS operating mechanism that controls access to memory and privileged instructions by a series of process levels or rings.

Scheduler

The part of the operating system that determines the order in which runnable tasks will be given control of the processor.

Security logging facility

A facility that allows the VS system to record security-related system events in a log file. These events can be captured in a report by using the LOGPRINT utility.

Semaphore

A variable stored in memory that is used to synchronize parallel processes. When a process is said to be waiting on a semaphore, it means that the process is waiting for another process to complete some activity.

Shared subroutine library

A single VS file that contains subroutines which are selectively accessed at runtime as opposed to linktime.

Stack

An area in memory that stores data items in consecutive locations. The stack is used to hold temporary items in a program and to hold linkage information.

Subroutine

A sequenced set of statements that may be used in one or more computer programs and at one or more points in a computer program.

SVC

A VS instruction that interrupts the program being executed and passes control to the operating system which performs the service indicated by the instruction.

System service

Operating system code that provides a function for applications programs.

Task

An environment (which may or may not be associated with a workstation or user logon id) under which a program runs and for which resources are allocated.

User address space

The access that a user has to physical and virtual storage.

Virtual address space

In virtual storage systems, the virtual storage assigned to a system task or a task initiated by a command.

Virtual region

A subdivision of the dynamic area that is allocated (in segment-size blocks) to a job step or a system task.

Virtual storage

The ability of the VS to address a storage space much larger than that available in physical memory.

Volume

A portion of a single unit of storage that is accessible to a single read/write mechanism, for example a disk pack or a floppy disk.

Volume table of contents (VTOC)

A table stored on a direct access volume that describes each data set in the volume.

NOTES

NOTES

WANG

ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851
TEL. (617) 459-5000
TWX 710-343-6769, TELEX 94-7421

Printed in U.S.A.
715-0423
10-85