MUSIC/SP Administrator's Reference

Part V - Chapter 18 - 23 (AR_P5.cm PS)

Part V. MUSIC/SP Internals

Chapter 18. System Internals

MUSIC/SP's Main Storage

Figure 18.1 shows the relationship between MUSIC/SP's real storage and virtual storage layouts.

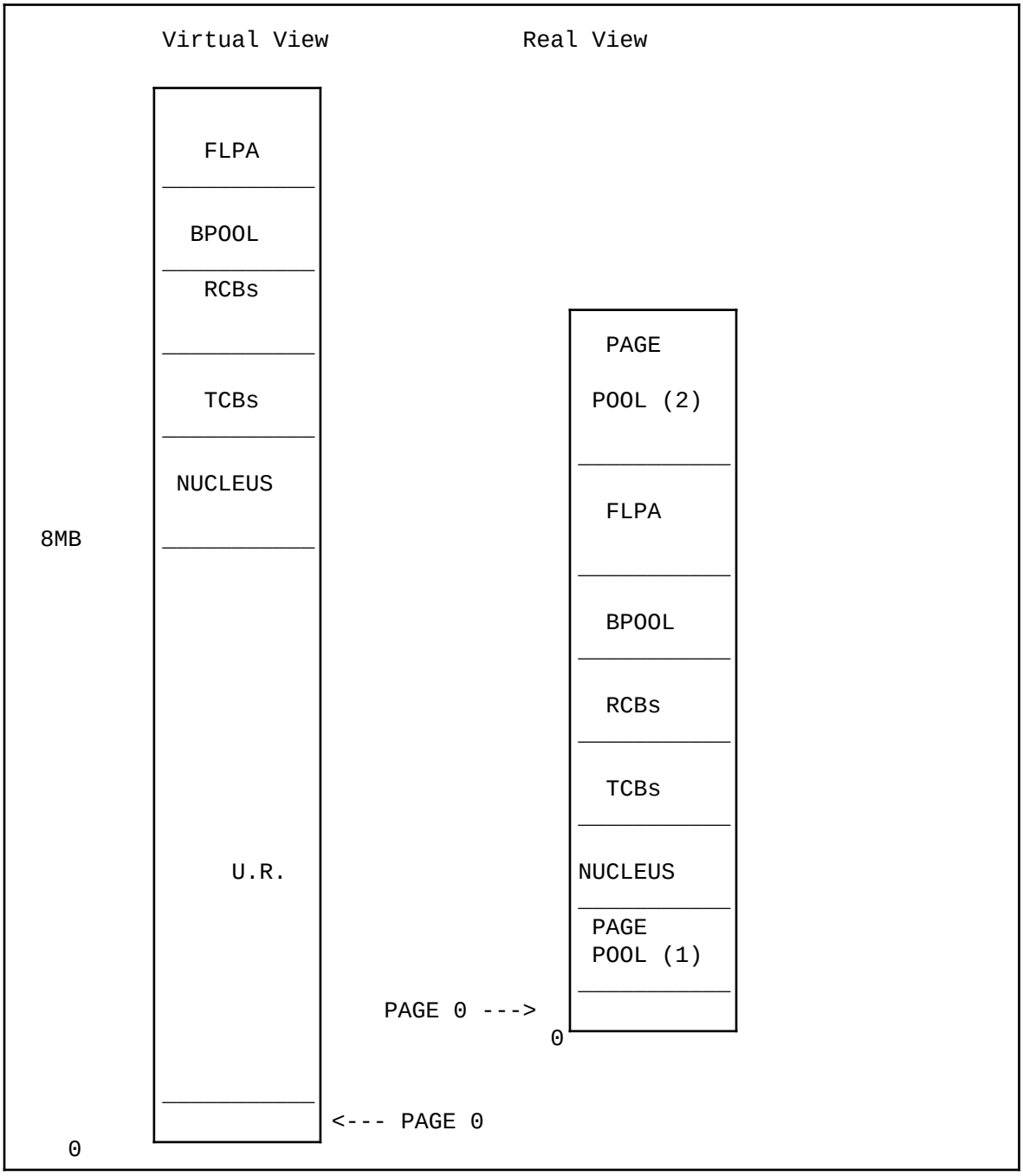


Figure 18.1 - Main Storage Layout of MUSIC

MUSIC/SP uses virtual storage. Page and segment tables are used to map the real storage layout to the

virtual storage layout. When MUSIC/SP is running the virtual storage layout is used by the majority of the system. The real storage layout is only of concern to those components of the system handling the page tables and low level I/O operations.

In the virtual view, the first 8 megabytes are taken up by an area called the User Region. Only a single user task can occupy this area at any given instant. The nucleus starts at address 8 megabytes and is followed by control blocks, buffers, and the Link Pack Area.

In the real view, the nucleus starts at address 30000 (hex). Thus any virtual addresses in the nucleus can be converted to a real address by subtracting 7D0000 (hex). The User Region component for each user is provided from the two page pool areas. At any instant only the pages for one user task are mapped to the virtual User Region area. If there is not enough space in these page pools to store the User Region pages for all user tasks, inactive tasks are paged or swapped to disk. The nucleus, control blocks, buffers and LPA are never paged or swapped.

Nucleus: contains the MUSIC/SP operating system supervisor program.

TCBs: contains the Terminal Control Blocks. There is one TCB for every terminal I/O device defined to MUSIC/SP.

RCBs: contains the Region Control Blocks. There is one RCB for every possible user job that is in the user region area.

BPOOL: contains the *Buffer Pool* (buffer pool) used to transfer information between the user regions and the terminals.

Fixed Link Pack Area:
contains high usage programs used by users. The use of this optional *FLPA* can improve MUSIC/SP's performance as you will see soon.

The User Region

The User Region itself has different sections as is illustrated in figure 18.2.

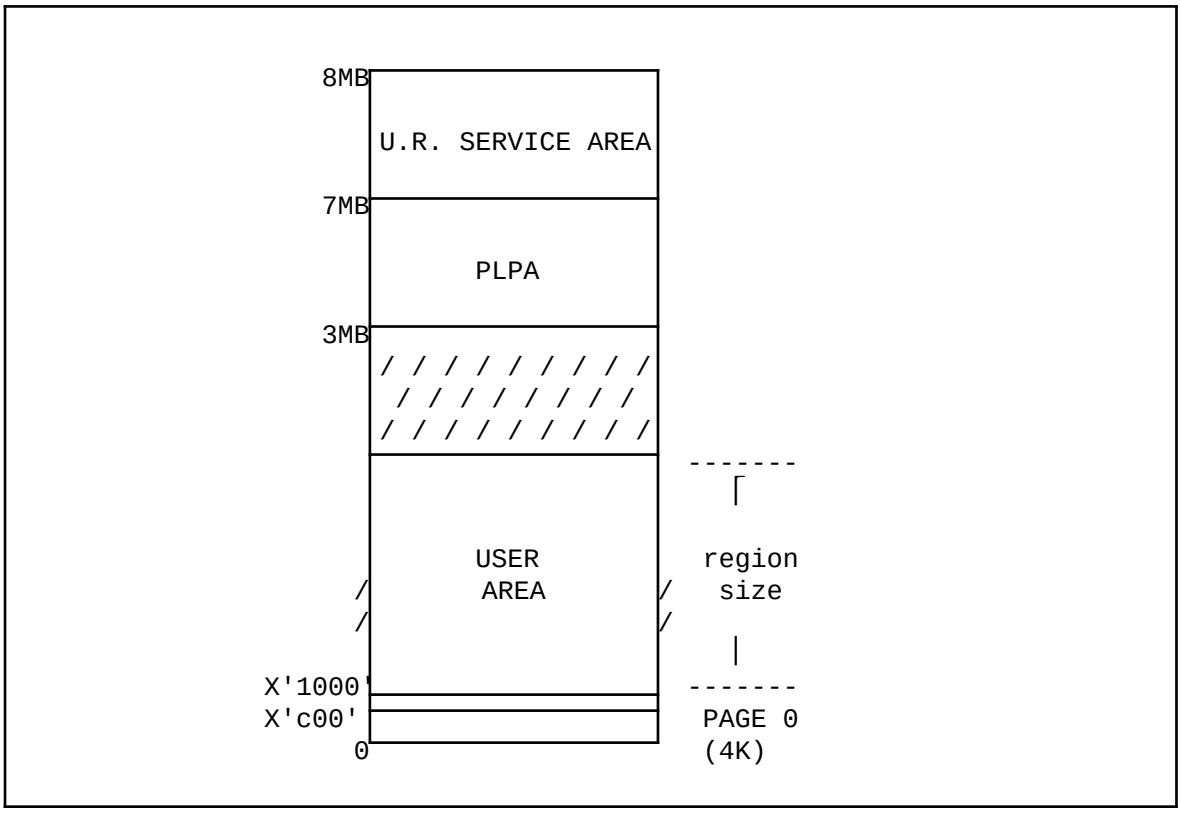


Figure 18.2 - User Region

User Program: This is where the users program is loaded. The maximum size of this area is 3 megabytes.

PLPA: The Pageable Link Pack area contains re-entrant modules that are called from programs in the user program area. This area extends from addresses 3 to 7 megabytes. The modules that can be loaded in this area have been preloaded on the systems page datasets during system initialization. This greatly reduces the loading overhead. Typically the PLPA contains compilers and system utilities that are commonly used.

Service Area: The User Region Service Area is used by the system supervisor in handling service requests from user tasks. It contains I/O buffers, save areas and stacks. One megabyte has been reserved for this area.

As mentioned above, not all users pages can be kept in memory at the same time. In addition, only one users pages can be mapped to the user region at any given time. A major goal of the system is to make sure that all the users get a fair share of the user region. There are three processes that work together to achieve this goal.

- The *scheduler* whose job it is to prioritize the requests for the user regions. The priority is based on two factors, one what kind of request it is and how long has the request been waiting. A job's priority is automatically set and readjusted by the system as the characteristics of the job changes.

- The *paging* process that attempts to keep only the most active pages of a job in main storage.

- The *swapping* process that can block page all the active pages of a job to auxiliary storage should the system determine that the storage can be better used by a higher priority job.

Nucleus Generation

The NUCGEN Utility is used to create the tape or punch file that when loaded writes MUSIC/SP's nucleus to disk. The output of the program is a sequential data set in card image format. The data set consists of the object decks for the MUSIC system and device configuration cards that define the physical I/O unit addresses for the terminals, disks, etc. These card images are preceded by a stand-alone loader utility.

The SYSGEN1 utility can be used to write the nucleus directly to disk while MUSIC is running. The following describes the process when this utility is not used.

The nucleus generation is initiated by an IPL from either the tape or the punch file produced by the NUCGEN utility. The IPL loads a small stand-alone loader utility. This loader then loads the object deck for the MUSIC system module SYGEN1 into main storage. SYGEN1 is a more powerful loader that uses disk work areas on the MUSIC IPL volume. SYGEN1 reads in and relocates the remaining MUSIC system modules. (Do not confuse this module SYGEN1 with the SYSGEN1 utility.)

Once the loading is complete, SYGEN1 moves a copy of the module storage map to the module HELLO. (This map is used by the main storage dump utilities.)

The module SYGEN2 gains control once the loading is complete. It reads the device configuration cards and builds device tables which will be used by HELLO when MUSIC itself is initialized.

SYGEN2 calls the module WMON to format and write the IPL information on the MUSIC SYSRES disk. Then WMON writes the nucleus in core-image format to the dataset SYS1.MUSIC.NUCLEUS on the MUSIC SYSRES disk. Control is then returned to SYGEN1 which writes a completion message to the console and enters the wait state. The new nucleus can now be used by IPLing from the SYSRES disk.

System Initialization

When the hardware LOAD function is performed, the MUSIC IPL text is read into main storage. This IPL text is located on track 0. It was written to disk by the module WMON during the nucleus generation procedure.

The module *MFETCH* reads in the MUSIC resident system from CKD disks and transfers to the initialization module *HELLO* which resides in what will become the user region. On FBA devices, the IPL text written on block 0 performs the function of MFETCH.

The following functions are among those performed. Main storage size and processing unit type are determined. Time and date are set. Free storage is zeroed and protection keys are set. Region Control Blocks (RCBs), Free page storage pool, I/O device Unit Control Blocks (UCBs) and Terminal Control Blocks (TCBs) are created. HELLO prints informative messages to the system operator concerning time, date and disk drive utilization. If any unusual conditions are found, appropriate error messages are issued and initialization is halted.

HELLO calls the module *DSINIT* to process the information in the system catalog. Its major functions are: Locate and read the system catalog, and allow the operator to modify it. Scan all ready disk packs, locate necessary system data sets and construct the Data Extent Blocks (DEBs), Pseudo Device Blocks and Statistics Blocks, positions the accounting dataset, load required modules into the Fixed Link Pack Area, and issue any VM commands found in the catalog.

When initialization is complete, the modules MFETCH, HELLO and DSINIT are no longer required and the

area they occupy is added to page pool 1. Control is then passed to the system scheduler/dispatcher module *URMON* to initialize servicing of user jobs. The first thing it does is run the module JOBONE. job.

JOBONE first scans the Save Library index and deletes any temporary files left around from the last time the system was run. JOBONE then loads the required modules from the load library into the PLPA (Pageable Link Pack Area). It loads each member into the virtual address specified on the RESPGM catalog entry which was read at IPL time. These addresses will be in the range X'300000' to X'6FFFFF'. Once loaded, each member is written out to the page data set. Users are not allowed to sign on to the system till JOBONE completes.

Interrupt Processing

The System/370 architecture defines five interrupt classes:

- Supervisor Call (SVC)
- Input/Output (I/O)
- External (EXT)
- Program Interrupts (PI)
- Machine Checks (MCK)

Supervisor Calls (SVC)

SVC interrupts are only caused by the SVC instruction. SVCs can be thought of as just a call to a routine. The caller need not be concerned with location of the routine in storage. SVC routines always start in the hardware Supervisor State with no I/O or External interrupts allowed.

When the hardware performs a SVC instruction it stores the current psw at storage location 20 (hex) and then loads a new psw from storage location 60 (hex). It will store a word at location 88 (hex) that contains the SVC number as the second half word.

When an SVC is issued control is transferred to the module *TRACE* which records the event in the system trace table. (This trace table is printed by the PRDUMP storage dump utility.) TRACE then branches to the module *SYSSVC*.

SYSSVC checks system status bits to determine if the SVC was issued in SYNC or ASYNC mode. If the SVC is issued by a system interrupt handler, it is in ASYNC mode. If the SVC is issued by a user program or by the system on behalf of a user program, it is said to be in SYNC mode. If in SYNC mode, transfer is made to the module *USRSVC* where the SVC is decoded and the appropriate service routines called to handle the request on behalf of the current user region. The user region service area (USRSVA) is used for buffers, save areas and the stack. Storage in the user region is directly addressable by the SVC service routines.

If in ASYNC mode, then SYSSVC handles the SVC itself. These are system requests and do not pertain to any particular user region, in fact the user region may not even be addressable at the time. Only a subset of the SVCs are valid in this mode.

Appendix C lists the SVCs that are allowed in sync and async modes.

SVC Handling in Trap Mode

There is another layer possible in the handling of SVCs. An user program can request control whenever an SVC is issued. This is known as TRAP mode. This mode is intended primarily for OS-Simulation, but is

also available for other applications. If an application is running in TRAP mode, then SVCs will cause control to be passed to an SVC processing routine that is part of the application. This SVC processor has no special supervisor powers as the system switches back to problem state with interrupts enabled before transferring control. The SVC is decoded and the appropriate action is taken to perform the required task.

For example, when a COBOL program is running, the COBOL object code issues OS SVCs. USRSVC will recognize that the program is running in trap mode and branch to the SVC handler OSTRAP which is in the user region. OSTRAP then may wish to issue an equivalent MUSIC SVC to perform the required function.

Most MUSIC SYNC mode SVC numbers are in the range of 126 and higher and the supported OS SVCs are from 0 to 125. This fact is used by the fast reflect SVC trap option. This fast reflect option is usually on in OSTRAP and so only the SVC numbers 0 through 125 return to OSTRAP for processing. So OSTRAP get control on OS style SVCs (0-125) but MUSIC's native SVCs (126-256) are handled directly by USRSVC.

I/O Interrupts

I/O interrupts are asynchronous interrupts. That is, they can occur at any time if the system mask in the current psw is enabled for I/O interrupts. This is the case when running user programs. I/O interrupts are not allowed while the system is processing SVCs on behalf of user programs except for those in OSTRAP.

When the hardware accepts an I/O interrupt it stores the current psw at storage location 38 (hex) and then loads a new psw from storage location 78 (hex). Location BA (hex) will contain the I/O address associated with the interrupt. The new I/O psw will cause the processing unit to transfer to the module *TRACE* which will record the event in the system trace table. (This trace table is printed by the PRDUMP storage dump utility.)

TRACE will check if the WAIT bit was on at the time of the interrupt and if so it will turn it off and update the wait statistical counters. (See the discussion under topic "Wait State" in this chapter for further details.)

TRACE will then branch to the module *DIOEX* that will check to see if the interrupt is for a disk or tape device. If so, it will update its queues and start more disk/tape I/O. (See the discussion under topic "Disk and Tape I/O" in this chapter for further details.)

If the interrupt is not for disk or tape it will branch to the module *MIOX*. MIOX checks if the interrupt is for a unit record device. Such devices are the card reader, card punch, batch line printer and the main system console. If so, it will process the interrupt.

If the interrupt is not for a unit record device, MIOX will branch to the module *TCS*. TCS handles the interrupts for the terminals. (See the discussion under topic "Terminal Handler" in this chapter for further details.)

External Interrupts (EXT)

MUSIC uses the external interrupts for the interval timer, the clock comparator and the CPU timer. Interfaces to VM services such as IUCV, VMCF, and Logical Device also use external interrupts. Interval timer interrupts are used only at system initialization time.

External interrupts are asynchronous interrupts. That is, they can occur at any time if the system mask in the current psw is enabled for external interrupts. These interrupts are enabled when the system is running user code.

When the hardware accepts an external interrupt it stores the current psw at storage location 18 (hex) and then loads a new psw from storage location 58 (hex). The halfword at 86 (hex) contains a code that identifies the kind of external interrupt. The new external psw will cause the processing unit to transfer to the module

TRACE which will record the event in the system trace table. (This trace table is printed by the PRDUMP storage dump utility.)

TRACE will check if the WAIT bit was on at the time of the interrupt and if so it will turn it off and update the wait statistical counters. (See the discussion under topic "Wait State" in this chapter for further details.)

TRACE will then branch to the module *URMON*.

Clock comparator interrupts are used only by the dispatcher. The clock comparator is set if the dispatcher found nothing to do at some point. When this interrupt comes through, the dispatcher will check its queues again. Outstanding clock comparator interrupts are not cleared if the dispatcher is entered for other reasons. Thus, it is possible to get these interrupts at a later time. Should this happen, they will be ignored.

CPU timer interrupts mean that a time slice has come to the end. URMON will then go to USRSVC to handle it. The transfer to USRSVC is handled as if an SVC 256 had occurred at the location pointed to by the external old psw.

When USRSVC processes the SVC 256, it will check to see if the job has used too much time. If so, then a bit will be set. The dispatcher is then called, in all cases, to end the current time slice.

When the job is dispatched again, and when it returns to problem state, it will check the bit to see if the job time has been exceeded. If so, a message will be printed and the job will be terminated.

Program Interrupts (PI)

Program interrupts (PIs) occur when the processing unit cannot perform an instruction because of (a) it cannot access one of the required storage pages or (b) it found bad data (such as division by 0) or (c) by the attempt to execute an instruction that is undefined or invalid.

Case (a) is known as a *page exception*. The PSW in this case will point to the actual instruction that caused the reference -- not the one following it as is the case with most other program interrupts. The hardware will store the virtual address of the page that could not be found at location 90 (hex). The low order 3 hex digits of this value cannot be counted on to be exact.

When the hardware detects a Program Interrupt it stores the current psw at storage location 28 (hex) and then loads a new psw from storage location 68 (hex). Location 8E (hex) contains the PI code.

This new program interrupt PSW will cause the processing unit to transfer to the module *PAGER*. This module checks to see if it was a paging exception. If so, then this module will resolve the paging exception if it can. Reference to a data page outside of the user region will cause the system to pass back a *protection check* interrupt. This will appear as PI code 4. Reference to an instruction on an odd address boundary that is outside of the region will be treated as *specification error*. This will appear as PI code 6.

If the PI was not be resolved as a paging exception then control is given to the module PITRAP. The action PITRAP takes depends on whether the PI occurred in Supervisor or Problem state. It determines this from the Problem State bit in the psw.

Supervisor State:	PITRAP will issue a console message and stop the system by entering a wait state with all interrupt masks off. A re-IPL of MUSIC is required to recover from this condition.
Problem State:	PITRAP enters the routine XERMON contained within PITRAP.
	XERMON contains routines to fix up the floating-point registers for program checks due to floating-point underflows or overflows.

If the user program has defined a program interrupt exit, then XERMON will branch to it. Otherwise the user program may terminate depending on the degree of severity of the error.

Machine Check Interrupt

Machine checks occur when the processing unit detects an error in its hardware. Examples include the detection of an unrecoverable parity error in storage. Channel checks may or may not cause a machine check interrupt, depending on the processing unit model. Channel checks that are not presented as machine checks are represented by certain bits on in the CSW presented with I/O interrupts. The module *DIOEX* checks for the presence of these channel check bits. If found, DIOEX will branch to the module MCHINT.

When the hardware detects a machine check condition, it stores the current psw at storage location 30 (hex) and then loads a new psw from storage location 70 (hex). This psw will cause the processing unit to transfer to the module *MCHINT*.

MCHINT logs the error. The action taken depends on the type of error:

Soft Machine Check: The system is allowed to continue if the error counts have not been exceeded.

Hard Machine Check: If the error can be localized to a specific user's job, then only that job is terminated. Otherwise, the system is shut down.

WAIT State

MUSIC enters a wait state when there is no processing unit work to do. (The wait state is indicated by the Wait bit on in the current PSW.) The following are the three main reasons for wait states on MUSIC.

- All available user region work has been performed. This is known as the MUSIC idle wait state. The last bytes of the wait psw will be FFFFFF in this case.
- All user regions are all in wait state. This usually means that each one has some I/O to do. The last bytes of the wait psw will be FF00FF in this case.
- Some system function cannot proceed until some I/O is completed. The last bytes of the wait psw will point to the virtual location of the wait request.

MUSIC provides a technique of recording how much time is spent in wait state and for what reason. A task that wishes to enter the wait state issues the MUSIC wait SVC followed by a 2 byte statistical counter number. (The system macro LOOP generates the required sequence of instructions.)

The module *SYSSVC* processes this SVC request. It forms a psw with an instruction address set to retest the completion of the event. The psw also has the wait bit on. This psw is then loaded and the system enters the wait state.

Upon either an I/O or External interrupt, the old psw is inspected in the module TRACE to see if the wait bit is on. If on, the bit is turned off and the time that the system was in wait state is recorded for statistical purposes.

After the system processes the interrupt, it will reload the old psw and thus resume what it was previously doing. If it was in wait state before, the old psw now has the wait bit off which will cause it to retest the completion of the event. If the event is now complete, the task will continue, otherwise the wait SVC will be

redone.

The WAITS utility program can be used to print the time information gathered by this process.

Job Dispatching and Scheduling

Scheduler

Requests for user region service are processed by the scheduler which is located in the module *URMON*. For example, when a terminal user enters a /ID command, TCS issues a $QUEIT SVC which is processed by URMON. Depending on the type of request, URMON then places the request into one of some 9 queues. These $QUEIT requests can come in at any time and do not have any immediate effect on the job currently running in any region.

The items within each queue are serviced in a first in, first out (FIFO) basis. That is, they will be taken in the order they requested service. The only exception to this is the futures queue which is ordered by the time the user wants service. The futures queue is entered by a call to the DELAY subroutine, etc.

Each queue has a bias number which determines the relative priority between queues. Take for example the following case. Suppose the conversational read queue has a bias of 10 and the full time slice queue has a bias of 200. The system would attempt to give response times in that ratio if reads were getting 0.01 second, then full time slice ones would get 0.2 seconds.

Installations can change the bias numbers by changing the table in URMON. These numbers can also be changed dynamically.

Dispatcher

The module *URMON* contains the MUSIC dispatcher. Its job is to keep the user regions busy. It does that by managing chains of Region Control Blocks (RCBs).

At IPL time the system builds a number of RCBs and chains them all into the free chain. Also at IPL time, the system establishes the maximum multiprocessing level (MAXMPL). This MAXMPL is the maximum number of RCBs that can be simultaneously in the active chain.

If any queue has some work that can be done, the system will try to add another RCB to the active chain if the MAXMPL has not been exceeded.

An RCB will stay in the active until one of the following happens to the job represented by the RCB:

- The job exceeds its processing unit time slice. This time slice is determined at IPL time. It is approximately one second divided by the MAXMPL number. Note that this time is pure processing unit time not elapsed time. The RCB is then placed in the dormant chain. The scheduler is called to add the job to the full time slice queue.

- The job exceeds its I/O time slice. This means that the job has performed more than 50 I/O operations in this time slice. The I/O count includes paging I/O. The RCB is then placed in the dormant chain. The scheduler is called to add the job to the full time slice queue.

- The job has ended. The RCB is then added to the free chain.

- The job requests input from the terminal. The RCB is then placed in the dormant chain. (TCS will call

the scheduler when this job is to be activated again.)

- The job has issued a tape rewind or other similar condition that specifically called the dispatcher to dispatch someone else. The RCB is then placed in the dormant chain.

Swapping and Paging

Region Control Blocks

The region control blocks (RCBs) contain information about the user regions that are currently in main storage. In addition to status information the RCBs contain I/O control areas and the page and segment tables. Associated with the page tables are the page frames that are mapped to the user region. An RCB can be active or dormant.

Active RCBs are those that are currently being given a time slice. The dispatcher manages the competition between these tasks for CPU and I/O service. The maximum number of active RCBs that you can have at one time is limited by the "Maximum Multi-Programming Level" (MAXMPL). This number is set during system initialization based on the amount of free storage. It is normally between 3 and 10.

When the time slice ends the RCB becomes dormant. The dormant RCB and any pages associated with it remain in main storage until the system needs pages to satisfy the requests of an active RCB. At this time the module *SWAPER* will be called to free up some pages. SWAPER will call URMON to scan through the list of dormant RCBs and select the one that it thinks will not be needed for a while. SWAPER will then write all the main storage resident pages associated with that job to the swap data set(s). It will write a copy of the contents of the RCB as part of the swap data. After the swap out operation is completed then the RCB can be added to the free chain and the pages added to the free page list.

When the scheduler decides to continue a job, it will check to see if its RCB is in the dormant chain. If so, it can be reactivated by simply adding the RCB to the active queue again. If the RCB cannot be found then it must have been swapped to the swap data set(s). In this case, the system must find a free RCB and sufficient free pages to hold the job again. This may require calling the SWAPER module to swap someone else out.

Maximum Real Region Size (MAXRRS)

The MAXRRS parameter limits the real storage available to a user task. The default value is 272K, but this can be overridden in the NUCGEN job. The value of MAXRRS can have a large effect on performance. If the storage requirements of a program are larger than MAXRRS, MUSIC/SP provides the required virtual storage using both demand and block paging techniques. The larger MAXRRS, the less paging is required. It is not a good idea to make MAXRRS too large however, since it can increase the swap load. The ideal choice for MAXRRS depends on the job mix, the available main storage and the disk and channel facilities.

Swapping

The swap data set is formatted into 4K blocks. It is suballocated in terms of multiples of 10 blocks. Each of these 40K areas is known as a *Swap Set*. By default the MAXRRS value is 272K. Adding the 8K for the RCB gives a maximum of 280K that must be swapped. This will take 7 swap sets. SWAPER will attempt to distribute them over the defined swap data sets. (SYS1.MUSIC.SWAPx). Since multiple swap data sets are usually on separate channels, this will allow for overlapping of the time to do one swap set on one channel with the time to do one on another.

Paging

Paging does not directly get involved in job scheduling or swapping operations. No matter how big the user's virtual storage size may be, the real storage allocated will never exceed MAXRRS. Once the MAXRRS limit is reached the task will page. The paging algorithm is designed so that a task will only page against itself and never take storage away from other competing tasks. This control has the following benefits:

- A job with large virtual storage demands will not overwhelm the system.
- It limits the amount of swap I/O that has to be done even for large jobs.
- It tends to smooth all user's response times by not being so dependent on the number or characteristics of other simultaneously running jobs.

At job start time, no pages are assigned. The job control information including the page tables is stored in the RCB. Pages will be assigned as page exceptions occur. The pages will be obtained from the system-wide page pool. Up to MAXRRS of pages can be obtained per job from this pool.

After the MAXRRS limit has been exceeded, the PAGER module will scan the assigned pages to decide which ones have not been used for a while. The state of the inactive page determines what to do next. The choices are:

- The inactive page is entirely filled with binary zeros so it will be immediately added to the free page pool.
- A copy of the inactive page is also on the paging data set and thus it does not have to be written to disk. This allows the system to be able to immediately add it to the free page pool.
- The inactive page cannot be reused until a copy of its contents are written to disk. It is removed from the active page mapping and is added to the list of pages that must be written to disk on behalf of this user but is not added to the free page pool. If the list has grown to 8 pages then write all 8 in one I/O operation and when done add all 8 pages to the free page pool.

The blocking of page-out operations is called *block paging* and is a significant performance benefit over performing a separate I/O operation for each.

Page-in operations are not blocked. This results in a better usage of the available main storage page frames.

PLPA members are loaded by simply updating the page tables in the RCB. When a specific PLPA page is referenced, then the system will use this information to read in that page from the page data set.

The paging system gets informed of specific MUSIC requests to zero storage. This includes deleting PLPA members and releasing storage at the end of processing phases such as compilers and loaders. Pages that get zeroed in these ways are immediately added to the free page pool.

Notice that users page against themselves and not against each other. This results in a more consistent response pattern. Also note that no paging is done if the user jobs never exceeds MAXRRS. The MUSIC editor only needs about 80K to run and so it does not normally page.

Terminal Handler

The module *TCS* supervises all terminal I/O in MUSIC. When no program is running (*Go mode) TCS reads a command from the terminal. With the exception of trivial tasks, executing the command usually involves running a program in the user region. TCS schedules the user region program and waits for terminal output or requests for input from the program. When the job has finished and all the output has been processed the user is again prompted for a command.

MUSIC uses data spooling techniques to reduce overhead in communicating with terminals. For example, when a program writes to a terminal, the data is simply moved into a buffer and the program allowed to continue as if the I/O had taken place. The module SIOCS is responsible for placing the data into these spool buffers. The terminal handler (TCS) fetches data from this spool and performs the physical I/O at whatever rate the terminal can accept. Automatic synchronization is performed when input is required and at end of job. The system informs TCS that data is pending on the spool by calling the routine WAKEUP. TCS then fetches the data from the spool, one record at a time, and processes it according to it's type. There are three basic types of spool records. The most common type contains output data which is sent to the terminal with the appropriate control characters. The second type of spool record is used to set various terminal options flags. These records are usually the result of calls to the $SETOPT SVC. The third type of spool record is used to initiate a read from the terminal.

TCS is entered on a terminal I/O interrupt (IOTRAP) or from a user region request (WAKEUP). If TCS is busy processing another terminal, the request is queued.

TCS calls several modules to perform part of the terminal support. The modules are:

TERMIO — to perform device dependent functions such as CCW construction and I/O interrupt handling for 2741, 1050 and TTY terminals.

TM3270 — to perform device dependent functions such as CCW construction and I/O interrupt handling for 3270s.

TRMCTL — contains control blocks specifying many parameters which control the way in which the system communicates with remote terminals. Many parameters, such as line length, carriage return rate, tab and backspace characters can be specified for many terminal types.

TCS refers to the information in this module by an index stored in the TCB. This index is set up by the SIGNON phase at /ID time based on the trmcls specification either given on the command or extracted from the user's userid code profile or the default one for the terminal type.

COMAND — Processes the control commands entered by the terminal users.

CHKPFK — is called by TM3270 to perform operations associated with program function keys such as multi-session requests, the retrieve function, and user defined operations.

DEFPFK — processes the /DEFINE command.

NEWTCB — manages the control block pool for multi-session support and provides the logic to add, delete, or move to a new session.

DSPOOL — provides access to the input and output spool and the buffer pool.

FSIO — serves as the interface between the terminal handler and the user region for full screen I/O requests.

TRANTB	contains all terminal translate tables except those used for 3270 APL.
APLTRN	contains the translate tables used for the 3270 APL support.
TABSET	inserts tab characters and idle characters in lines to be sent to a terminal. It uses the terminal's current tabs to ensure that the time needed to print program output is minimized.
TABCMD	processes /TABIN and /TABOUT commands entered from terminals. It translates the values on the commands into internal form and places them into the appropriate TCB.

Terminal Buffer Pool

During system initialization, an area of storage is allocated as the terminal buffer pool based on the BPOOL parameter specified in the NUCGEN. Each buffer in this pool has a 512 byte data area and a 4 byte pointer used in buffer chaining. With the exception of a small input area associated with each terminal, all terminal buffers space is acquired dynamically from this pool. The module DSPOOL and its entry points provide two levels of access to this pool.

The DSPUT and DSGET routines allow the caller to create and access chains of logical records in these buffers. (In-core spool files). Two such chains are provided for each terminal, the input and output chains. Terminal output, spooled input, and full screen I/O, transfer data between TCS and the user region using this type of access.

The GETBUF and FREEBUF routines allow the caller free access to the buffer pool without imposing the logical record format. The 3270 screen manager, conversational read, and the /DEFINE command use this type of access. The FREEBUF routine is also used to free entire buffer chains when a job is cancelled or abnormally terminates.

Since the buffer pool is a limited resource, users are limited as to the number of buffers they can own at one time. When a user exceeds this limit, the user's job is temporarily stopped until more buffers become available. This limit is dynamically adjusted so that it is reduced as the number of available buffers is reduced. It is possible for the system to run out of buffers. When this happens system performance will be effected since all operations that require the buffer pool will have to wait for buffers to become available and almost all terminal I/O requires the buffer pool.

Allocate at least four buffers for each active terminal. The COUNTS and BPOOL utilities can be used to monitor buffer pool usage. If the total number of buffers used consistently approaches the number allocated it is recommended that more buffers be allocated. If the total number used is consistently much less than you have allocated the storage is being wasted. As noted above, it is normal for the user limit exceeded count to be non-zero.

Disk and Tape I/O

The module *DIOEX* supervises the MUSIC disk and tape I/O. DIOEX handles the actual I/O execution on the selector channel(s). It will handle tape and disk, read and write requests, and special purpose functions.

DIOEX is entered by an SVC when a disk/tape I/O operation is required. It will validate the UCB number given in the caller's argument list. If the UCB number is greater than the highest disk/tape UCB, DIOEX will check if it corresponds to a valid Data Extent Block (DEB) number. (Most DIOEX requests use these DEB numbers.) (The DEB's are created at MUSIC IPL time by DSINIT with the exception of those used to

access User Data Set files. UDS DEB's are contained in the XFCB module and are initialized when the user job starts.)

Once the request has been validated, DIOEX adds the request to the channel queue depending on what physical channel the disk/tape is on. If the channel is free, DIOEX will see if it needs to construct a channel program. Most requests require that disk channel programs be constructed using information from the caller's request block and the appropriate Unit Control Block (UCB) and DEB. When the channel program is constructed, DIOEX will start the I/O.

DIOEX also performs the I/O interrupt handling for selector channels, including error detection, correction and logging. If the I/O request signals that the I/O is complete, DIOEX will update the channel queue and process the next request in the queue.

If an I/O error is indicated, a retry operation will be attempted.

The module *UIO* preprocesses all disk and tape I/O coming from user regions. It performs many of the functions that DIOEX does such as looking up DEBs. The CCWs prepared by this module use indirect addressing to access the real storage pages involved in the I/O. It might have to call the PAGER module to page-in part of a buffer. When the CCWs are prepared, URIO will call DIOEX to schedule the I/O. It will then call the dispatcher to run another user until the I/O is complete. When the I/O is complete, the original task will be redispatched and will then return to URIO so it can check the ending conditions and unlock the pages involved in the I/O.

User Region I/O

Most user region I/O requests result in an SVC that is handled by the module *MFIO*. Full details of the MFIO SVC interface can be found in *Chapter 20 - File System*. This single SVC provides the interface to the file system, the UDS file system, terminal input and output, tape, printers, punches and card readers. There are two basic ways that an application can access files and I/O devices.

Applications can access files dynamically. When the open request is issued, MFIO returns an *Internal Unit Number* that the application uses on all subsequent I/O requests. Some devices have unit numbers pre-defined and need not be specifically opened.

The /FILE statement is used to define a file or I/O device for the duration of a job. In this case, the module CTL opens the file and saves the internal unit number in one of two tables in page zero of the user region. The application specifies either a *DDNAME* or *Logical Unit Number* when accessing the file. The system translates this to an internal unit number using the tables set up by CTL.

I/O requests can specify either the internal unit number (1-255) or the logical unit number (1 - 15). Since the logical unit number corresponds to a /FILE statement, the actual physical device can be changed by changing the definition on the /FILE statement. Thus a program that reads from the terminal keyboard using logical unit 9 can be changed to read from a tape by adding a /FILE statement that defines unit 9 as a tape. A program using internal unit 9 will always read from the keyboard.

Unit numbers less than 16 are assigned to specific I/O devices or files. Unit number greater than 16 are assigned when files are opened by MFIO dynamically. The following describes the handling of internal unit numbers. Numbers 1 through 10 also correspond to the default logical unit numbers.

Units 1,2,3,4 These unit numbers are used to access the User Data Sets and tapes. MFIO calls the module *FIOCS* to process this request. FIOCS uses the control blocks in the module *XFCB* to buffer and control these four units. ($CTL, the execution start phase, processes the user /FILE cards and initializes fields in the XFCBs. The XFCBs are part of the module URSRVA.)

Unit 5 This is the system input unit. Commands, programs and input data are read from this unit. MFIO calls the module *SIOCS* to process the I/O. SIOCS starts out reading the terminal or batch input spool. If a /INCLUDE statement is encountered it switches to reading the file named on the /INCLUDE statement.

Unit 6 This is the system output unit. Program output and messages are by default sent to this unit. MFIO calls the module SIOCS to handle the I/O. SIOCS writes the output to the terminal or batch output spool. On batch the output is sent to the printer. On a terminal the output is displayed on the terminal.

Unit 7 This unit is the system punch and it is rarely used these day. On batch it represents the card punch, on terminals the paper tape punch. Again SIOCS is called to do the I/O and the data is written to the batch or terminal output spool.

Unit 9 This is used when a program wants to read interactively from the terminal. On batch it is treated like unit 5. Again MFIO calls SIOCS. SIOCS writes a special record to the terminal output spool and then calls *URMON* to end the time slice. When the module that handles terminal (TCS) detects this special record, it reads the input from the terminal and reschedules the program execution. Once the program regains control, MFIO passes it the terminal input.

Unit 10 This is the known as the HOLDING file. Some compilers write object programs to this file. MFIO will check if this is the first I/O on unit 10 in this job. If this is the case then it will create a file called @HOLD.sss. (*sss* is the subcode.) Any previously existing file with this name will be purged. Subsequent writes to unit 10 just add records to this file.

Units 16-255 These unit numbers are assigned when a file is dynamically opened using the OPEN request of the MFIO SVC.

Batch Spooling

The module *SPAM* supervises the batch spooling in MUSIC. It logically consists of two processes: (1) reading cards and writing them to the batch spool area on disk and, (2) reading the batch output spool area and printing or punching the records.

Batch Input Spooling

SPAM constructs the CCWs to read from the card reader. It calls the module *MIOX* to initiate the I/O operation and perform most I/O error recovery. It calls the module *XTXT* to compress the card images into the same format as used with /INPUT files. (Refer to the topic "Batch Input File Format" for a description of the packing scheme.)

SPAM calls the module *IOBUFR* to buffer these compressed records into physical spool block format. IOBUFR will perform the actual disk request when a spool buffer fills.

When a /ID card is detected the module *CKCDE* is called to perform the userid (code) checking function. CKCDE calls the module *CDSRCH* to locate and read in the userid (code) record. (The job time and page limits, etc, are not processed at this time. They will be handled later by CTL when the job starts running.)

When SPAM reads a /END card, the input spool file is closed and the user region job scheduler is called to queue the job request.

When SPAM reads a /PAUSE card, a console message is sent and spooling continues. When a /END card is later detected, SPAM will check if the operator has responded to the PAUSE message yet. If not, the job will not be queued at this time.

Batch Output Spooling

SPAM calls IOBUFR to read the output spool disk data set. Each logical record is prefixed with a one byte output route code which indicates if the record is for the printer or punch. (The output records are in compressed form with trailing blanks removed.)

SPAM forms the required CCWs to print or punch the record and call MIOX to perform the actual I/O.

Batch Internal Reader

Jobs submitted to the internal reader get written to the system data set SYS1.MUSIC.SUBMIT. Control information exists in the first block of this file. The data set is divided into a number of sections (also called queues), each representing one of the internal reader classes (1, 2,...). A submitted job occupies one block in the appropriate queue.

The subroutine SBMJOB ($SUB:SBMJOB.S) places a job into the submit data set. It also initializes the submit data set, if necessary. The routine SBREAD in the module SPAM reads the job from the submit data set. The module CAR processes the /RDR console command, used by the operator to control the internal reader.

For more information, refer to the comments and coding in SPAM and $SUB:SBMSET.S.

Batch Input File Format

Lines entered through batch as card images (80-character records) are compressed and blocked prior to being written on disk. This packing conserves disk space and reduces data transfer time.

Each card image is compressed by removing high-order (trailing) blanks from the following 3 fields:

Field 1: positions 1 to 6
Field 2: positions 7 to 72
Field 3: positions 73 to 80

Four 1-byte binary counts are appended to the compressed text to enable the line to be unpacked. The final format is

```
C  L1  L2  Text  C
```

where:

C total length of packed line, including control bytes. The C at the end of each line is used by the /DELETE command.
L1 length of field 1 after high-order blanks removed.
L2 length of field 2 after high-order blanks removed.
Text fields 1 to 3 in packed format.

Example:

```
 Column 1      7                    73
        17     FORMAT(I5)           FORT0003
```

appears as

```
        24  2  10  17FORMAT(I5)FORT0003  24
```

A blank line is stored as 04000004 (hexadecimal, 4 bytes).

The compressed lines are blocked into 512-byte physical blocks. A line is not split over a physical block. The last line in a block is always followed by a zero byte to indicate end of buffer. Thus, each 512-byte block may contain from 6 to 128 lines, depending on the amount of information in each line.

Console I/O

The module *CAR* manages the I/O to the MUSIC system console.

CAR is entered by an SVC when a message to the console is requested. The console queue is updated and if possible, the I/O started.

CAR is entered from MIOX upon a console I/O interrupt. The interrupt may indicate the completion of some I/O activity. It could also indicate that the console REQUEST button has been pressed.

Accounting Log

The module *DICDOC* handles writes to the MUSIC accounting log. Records are written to the log by using an SVC. DICDOC buffers these records and does disk I/O when required through calls to DIOEX.

The ACTDMP utility program is used to read this detailed log and produce single accounting records for each session. The format of the detailed log records is described below.

SPAM calls DICDOC to write the information from the /ID card read from the card reader. It also calls DICDOC to write out details of a job's output such as how many lines were printed and how many cards were punched.

SIGNON issues an SVC that calls DICDOC to write information from the /ID commands processed. TCS calls DICDOC to record the number of characters transmitted to the terminal, information about /OFF commands.

URMON calls DICDOC to write end of job records that contain the total amount of time used in the user region.

Accounting Record Formats

The accounting data is stored on the system data set SYS1.MUSIC.ACCT. Each block on the file is 4096 bytes in length. The first block in the file contains pointers into the file. The remainder of the blocks contain the actual accounting records.

The accounting data is written block by block to the disk file by the system module DICDOC. The blocks are written in ascending block number order until all the blocks have been used. Subsequent blocks will be written to block numbers 2, 3, etc. The pointers in the first block contain information that will stop the system from destroying blocks which have not been processed by the ACTDMP utility.

Each block ends with a four byte sequence number. This number is always one greater than the last block written. A discontinuity in these numbers indicates the logical end of the file.

Several record formats are used to hold accounting information in the disk accounting file:

System Initialization Record

```
      Time of day      Date           Clock

K     H H M M    M M / D D / Y Y   C C C C
```

Notes:

1. *K* represents the character count of the record (all counts include themselves).

2. The *CCCC* represents the time of day clock which is in units of two's-complement three-hundredths of a second. The value is stored in 4 bytes.

/ID Record

```
      Logical  Job  Packed ID Line   Clock
CNT    Unit    Code

 K      U U   0 1   X X X - - - X   C C C C
```

General Accounting Information Record

```
      Logical  Job   Request
CNT    Unit    Code    Type     Clock

 K      U U     J       R     C C C C
```

Job Code

02xx Job Request(xx is Request Type--see below)
03 Job Selection
06 Job Cancel or Job Cutoff (see below)
07 End of Output (See below)
08 /OFF Record

Request Type (code 2 records only)

01 /RUN or /EXEC
02 /UPDATE
03 /SAVE
04 /PURGE
05 Jobone system initialization phase
06 /DISPLAY OR /LIST

07 /ID
08 /RENAME
09 /INPUT

End of Output Record (#7) (Terminal)

```
Cnt  Logical      # of    Clock
      Unit        chars

 K     UU    7    XXXX    CCCC
```

End of Output Record (#7) (Batch)

```
cnt             crds rd  crds pch lns prt tape, disk mounts

 K      0     7   XXXX    XXXX     XXXX    X   X
```

Job Cutoff (#6) Records

```
     Logical    Job   Cutoff   Info     Execution
CNT    Unit    Code    Code    Bytes      Time     Clock

 K     U U      6       C       P KK     T T T T   C C C C
```

Job Code

Same as for General Accounting Information record.

Cutoff Code

01 Job Overtime
02 Not used
03 Normal EOJ
04 Not used
05 Abnormal EOJ
06 System I/O Error
07 Abnormal End of phase
08 Not Used
09 Cancel

Info Bytes (#6 records only)

P is processor identification set by CTL.

KK is the number of 4K pages allocated to the job. Minimum is 27 representing 108K.

UDS Delete Record (#10) (See module PST for details)

```
Cnt  Cde         Cde cont'd  Vol   Ucb Type  Details

K    CC    A     CCCCC    VVVVVVV    U  .....
```

Shutdown Record

```
      Shutdown
CNT     Code      Clock

 K    FF FF FF   C C C C
```

Ordinarily the accounting records for terminals will be recorded in logical sequence. However, they are recorded in time sequence, i.e., in order of their occurrence in the system. This means that /ID records for other terminals can be recorded before another job occupying the processing unit is done. The end of output rcd (#7) for batch may occur before or after the end of job record or after the next /ID record. The #7 records for terminals can occur anytime.

Accounting Record Notes

1. The execution time portion of a job cutoff entry is in service units times 300. It is the total chargeable time for the job.

2. /ID's are placed in the Accounting File in packed input format.

3. The accounting records are packed into 4096-character blocks on disk. A record with CNT = 0 indicates end-of-buffer. The last four bytes contains a sequence number. This number is incremented by one for each subsequent record. A discontinuity of these numbers indicates the logical end of the file.

4. Number 6 records usually occur after number 3 records. The exception is that a number 1 record (/ID) can occur between a number 3 and a number 6.

User Code Table

The User Code Table contains the information required by MUSIC for each userid authorized to use the system. This information includes passwords, time limits, options, privileges, etc. (Userids on MUSIC are sometimes called the user's code.) The table resides in two system data sets: SYS1.MUSIC.CODINDX, which is a 2-level index, and SYS1.MUSIC.CODTABL, which contains the individual userid records. Each userid, consisting of 16 characters with trailing blanks, is represented by a code record, which is pointed to by an entry in the index. The index provides quick access to any record, and also allows records to be added and deleted easily.

An SVC ($GETCOD, processed by module CDSRCH) is used to get the code record for a given userid. The SVC searches the index for the userid, then reads the corresponding code record into main storage. For example, at sign-on time, the module SIGNON uses this SVC to verify the user's password and fill in fields in the user's Terminal Control Block (TCB). It is also used by module CKCDE for batch jobs, and by the CODUPD and PROFILE programs.

The Code Table Update utility (CODUPD) is a conversational program which adds, changes, displays, and deletes code records. Usage of CODUPD, along with the various code record fields, is fully described in the *Chapter 17 - System Utility Programs*. A restricted version of this program (PROFILE) enables users to change and inspect some of the fields in their code records, such as passwords, default time limit, auto-program name, etc. PROFILE usage is described in the *MUSIC/SP User's Reference Guide*.

The Code Table Dump/Restore utility, CODUMP, is used to dump the entire code table to tape or a sequential file, for backup purposes. It can also restore the code table from a dump. The restore function recreates the index, compressing it at the same time. CODUMP is described in the *Chapter 17 - System Utility*

Programs.

Source for CODUPD, PROFILE and CODUMP is stored under code $COD in the Save Library (or on the MUSIC source tape). Code $COD also contains some subroutines that are useful for scanning and modifying the code table. File $MCM:CODENTRY.M contains the macro that defines the code record contents.

Code Table Structure

The code table index (SYS1.MUSIC.CODINDX) contains the master index record (the first record of the data set) and the second level index records. The data set is referred to by Data Extent Block (DEB) number 234. Each record is normally 4096 bytes, but a shorter length may be used. The master index record is kept in main storage, for faster code lookup.

Data set SYS1.MUSIC.CODTABL contains the actual code table records, one for each userid, pointed to by entries in the index. It is referred to by DEB number 233. Each record is 512 bytes.

Both DEBs use a pseudo device format, with 10 logical tracks per logical cylinder and 40 (code table) or 11 (index) records per logical track. Use of a pseudo format does not affect the number of records per real track.

The userid is a 16-byte key that is used for indexing the code records. Each index record has one or more 18-byte entries, each entry being a userid followed by a 2-byte pointer. The entries are in increasing order by key, and (unless the record is full) the last entry is followed by 18 hex FF bytes. An entry in the master index record contains the highest key of an index record and the pointer is the number of that index record (the index records after the master are numbered 1,2,3...). An entry in an index record points to a code record (the records in the code data set are numbered 0,1,2,...).

The maximum number of code records is limited by the number n of entries per index record. The maximum number of code records is n*n (assuming that the code data set is big enough). Index records are normally 4096 bytes long, so n = 4096/18 = 227, and the biggest code table could hold 227*227 = 51529 records.

The CODUPD program keeps track of unused index and code records by means of bit maps, which it builds at the start of each CODUPD run. CODUPD is the only program allowed to add or delete code records. All other programs can only get or change records (but they must not change the 16-byte userid at the beginning of the code record).

To avoid wasted space in index records when adding a large group of consecutive codes or subcodes, they should be added in increasing sorted order, i.e. the ADD commands should be sorted by userid.

Code Search SVC

The $GETCOD supervisor call instruction is used to search the code table index for a given code/subcode and read the code record into main storage. The SVC can also be used to request information about the code table, and to mark the main storage copy of the master index record as invalid. For the calling sequence and return codes, refer to the source of system module CDSRCH (under code $SYS). Subroutines CDSVC, CDPRM and INVAL (source under code $COD) can be called from a Fortran program to issue the SVC.

Chapter 19. Direct Access Storage

Overview

Two classes of direct access storage devices are supported by MUSIC: count-key-data (CKD) devices such as 3375 and 3380, and fixed block architecture (FBA) devices such as 3310 and 3370. Data records on CKD packs are accessed by cylinder, track and record number, and the records may be of various lengths. Records on FBA packs are accessed by a single block number (0,1,2,...) and are all of fixed length 512 bytes.

All disk packs used by MUSIC are in the standard OS/VS format. That is, they use normal volume labels and volume table of contents (VTOC). MUSIC CKD packs differ from OS in that most data sets are preformatted, which allows for more efficient access techniques. Furthermore unallocated tracks on UDS packs are preformatted in anticipation of future allocation, as described later on in this chapter. Preformatting is not required on FBA volumes since all blocks are of fixed length.

For CKD packs, the VTOC on MUSIC's IPL pack must not be greater than one cylinder. Format-5 DSCBs on CKD disks are used to keep track of free space, and the free space entries are maintained in order by disk address.

The format-5 DSCBs on FBA disks are not used (there is a single dummy format-5).

For purposes of data set allocation, blocks on FBA devices are grouped into *logical tracks*. A logical track is defined by MUSIC as a set of 32 512-byte blocks, in which the block number of the first block is a multiple of 32. All MUSIC FBA data sets must start and end on logical track boundaries. MUSIC's allocation routines automatically adjust to these boundaries.

Data sets allocated by MUSIC fall into one of two categories: SDS and UDS.

System Data Sets (SDS) contain all system controlled data such as spooling and swap areas, accounting files, Save Libraries and system residence data sets. A system data set must occupy only one extent. A system data set has a name of the form SYS1.MUSIC.xxxxxxx, where *xxxxxxx* is from 1 to 7 characters. The MUSIC/SP file system, the Save Library, is contained within these data sets.

User Data Sets (UDS) can be created by users by means of a /FILE statement. A UDS may occupy from 1 to 5 extents. A UDS has a name of the form ccccxxxx, where *cccc* is the 4-character sign-on code of the owner and *xxxx* is from 1 to 4 characters. Despite the name, most users never use User Data Sets. The MUSIC/SP Save Library is where most user files are stored.

These two types of data sets can be intermixed on the same physical pack. However, to allow for easier dump/restore operations, it is recommended that they not be intermixed. If user allocation is allowed on SDS volumes on CKD disks, then you must ensure that all the free space on the pack is preformatted to conform with the UDS requirements (ie. 512-byte records, no key). For example, before deleting a system data set, use the FORMAT utility to format its space to 512-byte records. The FMFREE utility can be used to format all the free space on a CKD volume.

All SDS packs are permanently mounted at IPL time. UDS packs normally are permanently mounted but batch jobs may make use of temporarily mounted disk packs.

A UDS CKD pack must be pre-formatted using the MUSIC disk format utility before it can be used for user data sets.

A SDS pack is not normally pre-formatted, but any data set allocated on the pack must be formatted prior to

its use.

At IPL time, the system initialization routine reads the system catalog (a data set on the IPL volume). This catalog provides (among other things), the data set names of all system data sets to be used. The volume label of the pack on which each SDS is to be found may also be listed. All ready disk packs are scanned. All SDS are located and control blocks are constructed to later allow system routines to use them. Any pack on which a SDS is located is marked as a SYSTEM pack. All other disk packs are marked as UDS packs.

System Data Sets

The following is a discussion of each type of SDS. This topic is useful if the installation wishes to expand or optimize these data sets for the configuration. Unless otherwise mentioned, the following rules should be assumed.

1. A SDS name must begin with the three characters SYS. The usual naming convention is to start a SDS name with SYS1.MUSIC.. The names used on the distribution system are shown at the beginning of each section.
2. The block size specified for each data set *must* be used. It cannot be altered. If the wrong block size is used, system errors may occur.
3. On the first line of each description, following the data set name, is the following information:

   ```
   data-set-name       identification,  block-size
   ```

   The identification and data set name are specified in the catalog. Certain data sets are not listed in the catalog. For these, no identification is shown.
4. All data sets listed are needed for MUSIC operation.
5. All MUSIC SDS must be one extent as multiple extent SDS data sets are not allowed.

Save Library Index

SYS1.MUSIC.UIDX **U000,512**

This data set contains the index of the MUSIC Save Library. Each file in the library is located through as reference to the data set. The usual size for this data set is 15552 records.

Save Library Space

SYS1.MUSIC.ULnn **Unnn,512**

This type of data set contains MUSIC files. They are numbered starting from 01 up to a maximum of 180. Each data set is referred to as a Library space. The *nn* above refers to the space number. At least one must be present. The starter system comes with several defined. New library volumes must be added in numeric order. Once a volume is added to the Library, it becomes part of the set and there is no quick way to remove it. Each library space can be up to 114,504 records in size.

Batch Input Spooling

SYS1.MUSIC.BATCHIN **B001,512**

This data set is used for input spooling for batch card reader/printer jobs. If it is not available to the system, batch jobs can not be processed. The number of records allocated to this file should be at least one sixth the number of cards to be read in for any single job. In practice, more than six cards will normally be placed in each record. The maximum number of records which will be used is 32767.

Batch Output Spooling

SYS1.MUSIC.BATCHOT **B002,512**

This data set is used for both printer and punch output spooling for batch jobs. It should be available if batch jobs are to run. Output lines are stored in compressed format with trailing blanks removed. A single record may hold several lines. The output from any executing programs may exceed the size of the data set. In this case, the job is temporarily suspended until all output already spooled to disk is printed or punched. Then the job is continued, re-using the spooling area. The maximum number of records of this data set that will be used by the system is 32767.

User Region Swap 1

SYS1.MUSIC.SWAP1 **F203,4096**

This data set is used to store user jobs while they are being executed and not resident in main storage. The block size used is always 4096 bytes. The system divides the blocks into groups of 10. Each group is called a *swap set*. A user's job can use up to nK of real main storage, where *n* is MAXRRS specified in NUCGEN. That size together with the RCB control block of 8K totals (n+8)K. Thus a job may require a total of (n+8)/40 swap sets. These swap sets can be divided up among the swap data sets SWAP1, SWAP2 and SWAP3. A job may use considerably less real storage, resulting in fewer space sets used. Each swap data set size can range from 10 to 64,000 blocks.

User Region Swap 2

SYS1.MUSIC.SWAP2 **F204,4096**

This data set and SYS1.MUSIC.SWAP3 are optional. They are of benefit only when each of the swap data sets are located on different disk channels. When this condition is met, the swap load is distributed among the channels such that they may be concurrently transferring parts of a single user's task. This can provide a significant performance enhancement. All the swap data sets should be approximately the same size, as the system will attempt to divide the swap load evenly between them and when it runs out of space of any one it will not be able to perform the swap operation.

User Region Swap 3

SYS1.MUSIC.SWAP3 **F205,4096**

See notes under SYS1.MUSIC.SWAP2 above.

Temporary User Data Sets

SYS1.MUSIC.SCRATCH **F206,512**

If a user job specifies UDS(&&TEMP) on a /FILE statements, this is an indication that the user wants a scratch file. This is one which is to be deleted at end of job. Instead of going to the trouble (and overhead) of allocating and then deleting a UDS, the system gives to the user a portion of this SDS. It in effect sub-allocates the UDS within this SDS. This data set is the exception to the normal SDS pack rule. It may reside on a normal UDS pack. A pack will not become a SYSTEM pack solely due to this data set. Of course, if any other SDS is found on the pack, it then becomes a SYSTEM pack. This data set can be up to 8176 tracks long. (On an FBA device, 1 track is considered to be 32 blocks.) Note that any job currently running may use this space and that any job may request up to 8,000 512-byte records of space (although most jobs do not use this much).

Page Data Set 1

SYS1.MUSIC.PAGE1 **F207,4096**

This data set is used to store infrequently used pages of user jobs while they are being executed. The block size used is always 4096 bytes. The system divides the blocks into groups of 8. Each group is called a *page set*. The maximum amount of page space that could be used per job depends on the maximum region the job has requested. This size is given on the /SYS REGION= statement associated with the job. Some additional storage beyond this size can also be paged. This holds items such as Save Library I/O buffers.

The first page data set must be big enough to hold all pages associated with the pageable link pack area modules. If your installation has fixed head devices such as a 2305, then it would be desirable to have this data set on it.

Each page data set size can range from 8 to 64,000 blocks.

Page Data Set 2

SYS1.MUSIC.PAGE2 **F208,4096**

This data set and SYS1.MUSIC.PAGE3 are optional. They are of benefit only when each of the page data sets are located on different disk channels. Unlike the swap data sets, the page data sets need not be the same size.

Page Data Set 3

SYS1.MUSIC.PAGE3 **F209,4096**

See notes under SYS1.MUSIC.PAGE2.

Batch Internal Reader

SYS1.MUSIC.SUBMIT **F225,512**

This data set is used with the internal batch reader support. Its minimum size is 200 blocks. A typical size is about 1800 blocks, which allows up to 250 jobs in queue 1 and up to 100 jobs in each of queues 2 to 16.

System Load Library

SYS1.MUSIC.LOADLIB **F226,2048**

This data set holds the processors and phases that may be loaded into the user region or link pack areas. It should contain at least 4000 records. Typical size is 12000 records. The LDLIBE utility program is used to initialize and maintain the load library. The directory should allow for at least 600 members.

Resident System Residence

SYS1.MUSIC.NUCLEUS **F227,note below**

This is the SDS in which the core-resident part of MUSIC resides. It is read into main storage at IPL time. The block size is 6400 for count-key-data devices and 512 for FBA devices. It should be at least 80 records long (1000 records if FBA device). It must reside on the IPL volume and its name must be as shown.

MUSIC Accounting File

SYS1.MUSIC.ACCT **F232,4096**

This data set is used to record all accounting data regarding computer and terminal usage. Records are entered here at such times as terminal sign-on and sign-off, job start and job end. The system will use whatever size data set is allocated. This data set must be initialized using the =RESET special IPL option before it can be used for the first time.

User Authorization Code Table

SYS1.MUSIC.CODTABL **F233,512**

This data set contains one record for each userid authorized to use the system. The maximum number of user code records in the code table depends on the block size B of the SYS1.MUSIC.CODINDX data set: the maximum number is (B/18)**2. Normally B is 4096, resulting in a maximum of 51529 codes or the number of records in SYS1.MUSIC.CODTABL, whichever is less. The total number of records in the data set must not exceed 52000. The number of records must be at least 40.

User Authorization Code Table Index

SYS1.MUSIC.CODINDX **F234,4096**

This data set contains the various index records which enable the system to quickly locate a specific code table entry. A completely full code table should require only (B/18)+2 records in this data set, where B is the block size of SYS1.MUSIC.CODINDX. However, due to fragmentation it is recommended that more records be allocated. This is particularly required when many additions and deletions are made to the code table. A recommended size is 1.5 * (B/18) records (i.e. about 340 records for the standard blocksize 4096). The Code Table Dump/Restore (CODUMP) utility

can be used to compress the code table to eliminate fragmentation.

The total number of records in the data set must not exceed 700. The number of records must be at least 11.

The standard and maximum block size of the index is B = 4096. A smaller block size may be used, but there is no advantage in doing so.

Spooling Area Status

SYS1.MUSIC.HPOOL -,6400

This data set is used by the system initialization at a system START to record the spooling areas and Save Libraries in use. It must reside on the IPL volume and its name must be as shown. It is only one track long.

Permanent Volume List

SYS1.MUSIC.HVLIST -,80

This data set is used to record the list of permanent volumes mounted at IPL time. Its name must be as shown, and it must reside on the IPL volume. The minimum number of records required is one plus the number of permanently mounted volumes.

System Data Set List

SYS1.MUSIC.DSLIST -,6400

This data set logs the data set names and volumes of all current system data sets in use at IPL time. It may be used by statistical and error logging or analysis routines. It is one record long, its name must be as shown, and it must reside on the IPL volume.

Generation Load Utility

SYS1.MUSIC.GENLOAD -,446

This data set is used during the time the system nucleus is being loaded on disk. It is not required during normal operation of MUSIC, but since it is rather small in size, it is recommended that it always be present. Its size is at least 100 records, its name must be as shown, and it must reside on the IPL volume of the generated system.

System Catalog

SYS1.MUSIC.CATALOG -,80

This data set is used to contain the MUSIC system catalog which describes the MUSIC SDS files and their location. This catalog is used only at MUSIC IPL time and must be located on the IPL volume. It should have at least 300 records.

Accessing Data Sets

Data on disk can be accessed in a variety of ways. The most basic method is to invoke the disk I/O scheduler passing the Unit Control Block (UCB) number of the disk to be used, and the absolute cylinder, head, and record of the data on the pack. This method is in general only used at system start-up time and by certain error and recovery routines.

The more common method is not to refer to the UCB, but to a Data Extent Block (DEB) number. There is one such number for every system data set used by the system. In addition to the DEB number, the routine indicates which record(s) of the data set it wishes to read or write. It may use a record number for this (first record of data set is number one, etc), or it may pass a pseudo-device CCHHR disk location. In the latter case, the program assumes that the data in the SDS is arranged as if it were a disk with a specific number of records per track, and heads per cylinder. It constructs what it considers the correct cylinder, head, record address. The disk scheduler converts this to a record number according to the pseudo-device format of the

data set (that is, the number of pseudo records/track and tracks/cyl.). The record number is then normally processed. Each data set which may be accessed in this way has a pseudo-device format specified in the catalog.

For more information about the System Catalog, refer to the EDTCAT utility in Chapter 17.

Allocating MUSIC System Data Sets

MUSIC system data sets are allocated in a similar manner to user data sets. However, several extra parameters are available on the /FILE statement.

All MUSIC system data set have data set names of the following form:

```
SYS1.MUSIC.xxxxxxx
```

The last identifier is a 1-7 character alphanumeric string, the first of which is alphabetic. Since the /FILE statement only allows a maximum of 8 characters in a data set name, the percent character (%) is used to abbreviate 'SYS1.MUSIC.'. A system data set is limited to one extent.

Instead of defining the size of the data set in number of records, the number of tracks is specified via the NTRK parameter. This parameter is only valid when the data set starts with the % character. The NREC parameter may not be used to allocate system data sets. For a data set on an FBA (fixed block architecture) device, one track is considered to be 32 512-byte physical blocks.

The BLK parameter must be used to specify the block size of the data set. For an FBA data set, this is the *logical* block size; a logical block is composed of one or more physical 512-byte blocks, and each logical block starts on a new physical block.

The MUSIC VIP code privilege is required to allocate or delete a system data set.

An example of a job to allocate a system data set follows. This will be a new batch input spooling data set. The BUFNO parameter avoids the unnecessary allocation of buffers in the user region.

```
/ID    appropriate parameters
/FILE 1 UDS(%BATCHIN) NTRK(250) BLK(512) VOL(MUSICX) NEW BUFNO(0)
/LOAD IEFBR
/END
```

Note that IEFBR is a dummy program which simply returns control to the system.

Extra parameters are available (although not often used). They allow a data set to be allocated at a specific absolute address on a disk pack. For example:

```
CC(cc) HH(hh)
```

The values in parenthesis are the starting cylinder and head addresses of the new data set (in decimal). For this allocation to be successful, sufficient free space must be available at the specified address. The 'LISTV' function of the DSKDMP utility may be used to inspect the position of free extents on disk packs. On FBA devices, specify cc as 0 and hh as the logical track number (the starting physical block number divided by 32).

The figures below are useful for calculating the number of tracks needed for data sets. Figures 19.1 and 19.2 give the track capacity (number of records) for various size records. Figures 19.3 - 19.5 review some disk

device characteristics.

RECORD SIZE (IN BYTES)	RECORDS PER TRACK 2314	3340	3330	2305M1	2305M2	3350 NAT
80	40	34	61	28	53	72
446	13	13	22	16	23	30
512	11	12	20	15	20	27
1024	6	7	11	10	12	15
2048	3	3	6	5	6	8
4096	1	2	3	3	3	4
6400	1	1	2	2	2	2
VTOC DSCBS	25	22	39	18	34	47

Figure 19.1 - Disk Device Track Capacity (*Part 1 or 2*)

RECORD SIZE (IN BYTES)	RECORDS PER TRACK 3375	3380	3390	9345		
80	75	83	78	67		
446	43	49	52	44		
512	40	46	49	41		
1024	25	31	33	28		
2048	14	18	21	17		
4096	8	10	12	10		
6400	5	6	8	6		
15476		3				
17600	2					
VTOC DSCBS	51	53	50	45		

Figure 19.2 - Disk Device Track Capacity (*Part 2 of 2*)

	2314	3340 M35	3340 M70	3330 M1,2	3330 M11	2305 M1	2305 M2	3350 NATIVE
Max BLKSIZ	7294	8368	8368	13030	13030	14136	14660	19069
Trks/Cyl	20	12	12	19	19	8	8	30
No. of Cyl	200	348	696	404	808	48	96	555
Capacity mb	29.1	35	70	100	200	5	11	317
Capacity mb Blksize 512	22.5	25.6	51.3	78.6	157	2.9	7.8	230
Capacity mb Blksize 4096	16.3	34.2	68.4	94.3	188	4.7	9.4	272
Xfer Rate kb	312	885	885	806	806	3000	1500	1200
Av Access ms	60	25	25	30	30	0	0	25
Rot Delay ms	12.5	10.1	10.1	8.4	8.4	2.5	5.0	8.3

Figure 19.3 - Disk Device Characteristics (*Part 1 of 3*)

	3375	3380 /arm	3380E /arm	3380K	3390	3390 .2	3390 .3	9345	9345 .2
Max BLKSIZ	35616	47476	47476	47476	56664	56664	56664	46546	46546
Trks/Cyl	12	15	15	15	15	15	15	15	15
No. of Cyl	959	885	1770	2655	1113	2226	3339	1440	2156
Capacity mb	409	630	1260	1890	946	1892	2838	1000	1500
Capacity mb Blksize 512	235	312	625	937	418	837	1256	453	678
Capacity mb Blksize 4096	377	543	1087	1631	820	1641	2461	884	1324
Xfer Rate kb	1859	3000	3000	3000	4200	4200	4200	4400	4400
Av Access ms	19	16	16	16	9.5	12.5	12.5	10	11
Rot Delay ms	10.1	8.3	8.3	8.3	7.1	7.1	7.1	5.6	5.6

Figure 19.4 - Disk Device Characteristics (*Part 2 of 3*)

	3310	3370-1 per arm	3370-2 per arm	9332	9335
Block Size	512	512	512	512	512
Capacity mb	64.5	285	364.9	184	412
Capacity bk	126016	558000	712752	360036	804714
Xfer Rate kb	1031	1859	1859	2500	3000
Av Access ms	27	20	19	19.5	18.0
Rot Delay ms	9.6	10.1	10.1	9.6	8.3

Figure 19.5 - Disk Device Characteristics (*Part 3 of 3*)

Adding Disks to MUSIC

All MUSIC packs use standard OS/VS type formats. That is, they have standard labels and VTOCS. This label and VTOC must be created before attempting to use a new disk on MUSIC. The Device Support

Facility program (DSF) can be used to perform this initialization. In addition, it is useful in detecting and correcting disk errors. A copy of DSF can be loaded by IPLing from the SOURCE tape. For more information see Device Support Facilities User's Guide and Reference (GC35-0033).

The OS/VS IEHDASDR or standalone IBCDASDI programs can be used to initialize non-FBA disks. The MUSIC utility INITFBA can be used to initialize FBA devices.

When initializing a pack any volume name may be used, as long as no two packs (to be used at the same time) have duplicate names. The user is cautioned against changing the name of the MUSIC1 UDS volume as certain execution procedures (for example PL/I) refer to it.

If a volume is to be used for User Data Sets, it should be formatted as documented under the Disk Format Utility (FORMAT). A VOLUME specification should be added to the MUSIC system catalog to enable users to allocate on this pack. All that is then required is to ensure that it is online when MUSIC is next loaded. In addition, the DSLIST file should be changed to include this new UDS pack (last statement of file). Take care to ensure that the file remains execute-only (XO). If the DSACT program is used, its control statements should also be changed to include the new volume.

If a volume is to be used for System Data Sets (SDS), it must be mounted and ready at IPL time. Any required data sets should be allocated as detailed in the previous topic of this chapter. The data sets should then be formatted using the Disk Format Utility (FORMAT).

The DSCOPY program can be used to move the contents of system data sets from one disk to another without using tape. The catalog statements in file SYSCAT should be modified to reflect the new data set locations. The catalog creation utility (EDTCAT) should be run to create a new catalog. The system should then be reloaded. If the code table was newly created, a current copy (dumped before reloading) should be restored using the code table dump/restore program. If a new subroutine library was allocated, it should be recreated by performing a subroutine library edit. If new Save Library data sets were allocated, formatted, and specified in the catalog, they are automatically incorporated into the Save Library at IPL time.

The case of moving existing Save Library data sets is a somewhat special case. Care must be taken to ensure that *nothing* on the entire Save Library changes from the time any of the data sets is copied until the time the system is loaded using this new library.

It is recommended that the starter system (after generation) be preserved so that in the case of any failure of the production system, a workable, compact system is still available.

Migrating MUSIC to a Different Type of Disk

Take the following steps to convert DASD.

1. Initialize the new volumes, using e.g. DSF as in the *MUSIC/SP Administration Guide*, "Installation" chapter. Configure your existing MUSIC system so that the new volumes are accessible to it, as extra volumes. You will probably need to assign different volume names to the new volumes for now, to avoid duplicate names -- e.g. MUSNWX, MUSNW1. You will rename the new volumes later, to their standard names MUSICX, MUSIC1.

2. Format the new volumes like UDS volumes, i.e. 512-byte blocks. Use FORMAT program with /INC FMTUDS. See control statements in description of FORMAT utility in MUSIC Admin. Reference. This job (and most jobs that follow), requires the /VIP ON command to be entered before you run it.

3. Allocate the new system data sets. These are all the SYS1.MUSIC.xxxxxxx data sets that MUSIC needs. See the chapter "Direct Access Storage" in the Admin. Reference. Use your existing data sets as a guide.

4. Format any system data sets which do NOT have blocksize 512, using the FORMAT program. They are SWAPn, PAGEn, LOADLIB, NUCLEUS, CODINDX, HPOOL, HVLIST, DSLIST, GENLOAD, CATALOG.

From this point on, keep all users off of MUSIC, and stop all BTRMs, until the migration is complete.

5. Use DSCOPY to copy any system data sets which contain data: UIDX, ULnn, SUBMIT, LOADLIB, CODTABL, CODINDX. (Do not copy NUCLEUS and CATALOG, since they will be generated below.)

6. Copy UDS files:

   Note: If your users do not use UDS files, or if you have only a few UDS files, you can skip this step and copy the UDS files manually, using DSCOPY.

   a. Scan old VTOC's (using STDSCB routine) to produce job stream of DSCOPY jobs, preserving LRECL, backup, and number of records.
   b. Temporarily REP the module PRE to not update the date of last access: DS1REFD field in format-1 DSCB, near label PREB106.
   c. Submit the DSCOPY jobstream to batch.
   d. Run a program to copy some format-1 DSCB fields from old VTOC to new:
      - date and time of creation
      - date and time of last accounting
      - date of last access
      - opened-for-write bit

7. Prepare the new system catalog (with ultimate volume names) and use EDTCAT program to write it to the new IPL volume.

8. Prepare a NUCGEN tape for the new system: new unit addresses, device types. This involves running your $GEN:NUCGEN.JOB file (updated as required), with output to tape. See the NUCGEN utility in the Admin. Reference.

9. Rename the new volumes to their final names, by using the NEWID option of the FORMAT utility. This needs /VIP ON.

10. Shutdown MUSIC and remove the old volumes.

11. IPL the new NUCGEN tape.

12. IPL the new system and test.

Chapter 20. File System

Save Library

MUSIC/SP's file system is called the *Save Library*. Physically the Save Library consists of a group of system data sets named SYS1.MUSIC.ULnn and an index data set called SYS1.MUSIC.UIDX. The system manages this common pool of disk space to provide file space for the users. Users need not be concerned about the physical organization of these data sets. Logically, each user has their own private set of files. The ownership of these files is based on an *ownership id*, which in most cases, is the userid that is used to logon to the system. The exception is for userids that have subcodes - the subcode is excluded for file ownership since these users share the same set of files.

File Names

A file name consists of a 1 to 17 character name. These file names can be stored in directories. These directories give the file system a tree-structured hierarchical structure.

The file system can refer to any file in the system by using what is called a "fully qualified" file name. All files in the system have a unique fully qualified name. An example of a fully qualified name is:

```
ABCD:CS100\NOTES
```

In the above example, ABCD refers to the ownership id. Note the colin (:) is used to separate the ownership id from the rest of the name. The file name is NOTES and it is located in the directory called CS100.

The file system has reserved 64 characters to hold the fully qualified name field. The directory and file name portion including any "\" characters can be up to 50 characters in length in total.

Usually users only use the *filename* part when referring to their own files. The ownership id and the colon can be prefixed in front of the directory and file name portions to refer to files belonging to other users. If a file has been stored with the share (SHR) attribute then any user can access that file. An administrator with the FILES privilege can access any file by prefixing file names with an ownership id.

Files can also placed in something called the *Common Library*. In this case a special index entry is created so that all users can reference the file without having to prefix the file name with the ownership id. Files for MUSIC/SP commands and utility program are in this Common Library. Files that are to be accessible to all users would normally be placed in this library. There is an option in the user profile that can be set to allow or disallow saving files in the Common Library.

Record Formats

The following record formats are supported (FC is the most used format):

F Fixed Format. All records are the same length and are physically stored on disk that way. Direct access is possible with such a file since each record occupies a fixed and predictable position on disk. The system maps the fixed format records to the physical 512 byte blocks. Records less than 512 in length are packed into the physical blocks but do not span those blocks. Records greater than 512 may span blocks but, each new record must start at the beginning of a physical block.

FC Fixed Compressed. To the application, all records appear to have the same record size. The record

size is defined when the file is created. Internally the file is compressed with repeated character sequences of four or more of the same character reduced to two bytes. Trailing blanks at the end of each record will never be written to disk. Logical records may span 512-byte disk boundaries though the user never needs be concerned about the internal representation of this file type.

V Variable Length. Each record is written to disk without compression. The length of each record is also written so that the length may be returned when it is read. The system keeps track of the size of the largest record written. The maximum record length is 32760. Trailing blanks at the end of each record may be truncated, by user option, so that they will never be written to disk. Logical records may span 512-byte disk boundaries though the user never needs be concerned about the internal representation of this file type.

VC Variable Compressed. Internally the file is compressed with repeated character sequences of four or more of the same character reduced to two bytes. The length of each record is also written so that the length may be returned when it is read. The system keeps track of the size of the largest record written. The maximum record length is 32760. Trailing blanks at the end of each record may be truncated, by user option, so that they will never be written to disk. Logical records may span 512-byte disk boundaries though the user never needs be concerned about the internal representation of this file type.

U Undefined. This format allows the user to use any style data handling desired, although the user must do the deblocking or blocking since the logical record routines will not attempt to handle it.

Record Size

The maximum record size is 32760 bytes. The minimum length is 0 for V and VC, and 1 for F and FC.

File Sizes

The minimum file size is four 512-byte records for a total of 2048 bytes.

The first 512-byte record of a file is used by the system to keep control information about the file. This record is called the *header*.

There is no absolute limit to the size of a file. A file's initial allocation must be satisfied in one library space. Since a single library space can hold up to 57 million bytes, that limits the initial size of a file. For any particular user, the system administrator can set a personal limit to be smaller than this.

Direct Access Support

Regardless of the format, records in all files can be accessed directly by telling the file system to position itself at a particular location in the file. This feature is particularly useful with fixed format files, where the application knows the positioning information in advance.

For all formats except U, file positioning information is given in the form of a relative byte address (RBA). The RBA is relative to the first block of the file. The RBA of the first usable byte in the file is 512 since the file's header occupies the first 512 bytes. The RBA of the second usable block in the file is 1024. Record format U files are positioned by physical block number.

The access methods supported by MUSIC allow direct access based on record number on fixed format files. OS-style NOTE and POINT operations can be done on all files excepting format U. A *Write Rule* is used to avoid the security exposure of accessing the old contents of file written by the previous user. (See Usage Notes for details.)

Dynamic Access

Files can be dynamically created, deleted, opened, closed, expanded and read or written into during the execution of a program.

RAS

Reliability and serviceability is enhanced through the following techniques:

The bitmaps, used to manage file space, are checked with a check-sum technique. When a purge operation is done, the system checks when bits are added back to the maps to detect the case of freeing space already freed. In this case, the total bit count is maintained correctly. If the checksum fails, the system will remove this space group from future allocations. The system will keep running.

Should a part of the library be unavailable, (due to failure of disk drive, for example), the system will still be able to operate. Attempt to use files in the unavailable space will result in a message to the user.

Should the master index get destroyed, it is possible to recreate it in most cases by scanning the library spaces since the file header has an unusual character sequence at the start of it. Headers of purged files are zeroed.

User Controls

Associated with each userid is a User Control Record (UCR) that controls the amount of file space that the user can have. Even privileged users cannot save a file under a userid that has no UCR record. The UCR record contains the maximum amount of space the user can allocate per file and maximum total space for all the user's files. The record also contains the amount of space currently used. All space amounts are expressed in units of K bytes (K=1024). (Since the basic allocation unit is currently set to 2K, the space used amounts will be even multiples of 2.)

Usage Dates

The system maintains three dates: date last opened for writing, date last opened for reading only, and the date the file's header was created. The time of day that the file was last opened for writing is also maintained.

Audit Information

With each file is maintained the full userid of the creator and last writer.

There is an option to maintain a usage count for each file recording the number of accesses.

Access Control

The basic options are PRIVATE and PUBLIC. A private file can be accessed only by its owner. Public files can be read by any user but only written by the owner. A user can select that a file can be added to but not read or written in any other way (APPEND). A file can be EXECUTE-ONLY, meaning that it can only be read by the systems job startup routines. User's can run the programs in these files but cannot look at them. The access control can differentiate between two classes of users: file owners and others. Thus a file can be execute only to others but not to the file owner. A number of combinations of access controls are available. The utility FILECH can be used to set or modify them.

Tag Field

Each file has a 64 byte tag field. This field can be used for any purpose the user wishes. It is not read or written as part of the data of the file but is part of the system information in the file's header. For example, it could be used to hold a phrase to describe the contents or usage of the file. Directory names can also contain a tag field.

Installation Field

Each file has a 16-byte installation dependent field that can be used for any purpose. It is reserved for use by installations other than McGill or for features that McGill wishes to use and never be part of the distributed MUSIC system.

Extendable

The Save Library index is a separate data set allowing it to be expanded or put on a faster device should that be desirable. This data set contains the hash numbers used with the index.

The number of concurrently open files can be expanded by the re-assembly of MFIO and MFIOCB. The space used per open file is 108 bytes. The number of open files bears no relation to the number of available buffers.

The size of the buffer pool used by the SL routines is also expandable. Each buffer requires 512 bytes.

Naming Conventions

Refer to the *MUSIC/SP Users Reference Guide* publication for a complete description of file naming conventions and directories. The following is a summary of the conventions.

File Names

The file name can be up to 17 characters in length. The user's current directory name is prefixed to the file name and can result in a string of up to 50 characters long. A "\" character will separate the directory prefix from the file name portion.

The user can prefix the userid (excluding subcode) and/or a directory to a file name. This is useful to refer to files owned by other users or to files in other directories. The general form of the fully qualified name is:

```
USERID:directory\filename
```

Character Set

The following characters are allowed in the ownership id and file name fields:

letters: A through Z (Upper Case Characters Only)
numbers: 0 through 9
special characters: the five (5) characters: # $ @ _ .

The special character "\" can be used when it follows the rules of the tree-structured directory naming

conventions.

The file name can include more special characters than ownership ids. They are:

```
- + % ! & ~
```

The first character of the file name cannot be any of these special charaters or a period (except for the special name of &&TEMP).

National Language File Suffix

File names can be suffixed by the 3 special character sequence "{@}" when used to open old files. This is used in the support of national languages. It allows programs to refer to different files depending on the user's language preference.

At open time, the system will replace these 3 characters with a single character taken from the national language option in effect. For example, if a user has selected the national language option of Portuguese, then opening with the character string "MSG{@}" will result in the attempt to open the file "MSGP". Should that file not exist, then an attempt will be made to open file "MSG". If the current language selected was English, then the file suffix is ignored. The 3-character file name suffix is not counted as part of the 17 character name although the generated name must be 17 characters or less.

The national language suffix characters are:

blank	English
F	French
K	Kanji (Japanese)
P	Portugeuse
E	Spanish
G	German

Reserved File Names

File names should not start with the commercial 'at' sign (@) since certain system utilities will use names of this form. The search order for files starting with this character is also different. For example:

@ADMIN.xxx	Used by ADMIN system facility
@ADMIN.CONFIG	Configuration file in the NUCGEN
@AMS.sub	Used by VSAM's AMS
@APLW.wsname	VS APL workspaces
@APLv.vvvvvv	Used to store VASPL external variables
@APL.DUMP.nn	Used to hold dump of VS APL
@AUTHSCHED	Used by SCHED
@CF.xxxx	Used to hold CONF xxxx
@CI.xxxx	Used by CONF
@CONF.sub	Used by CONF utility to store user's full name
@CONLOG	Used by CONLOG
@CW.sub	Used by Waterloo C
@ELOG.sub.nn	Editor restart file.
@GNJOB	Used by GENPCH
@HOLD.sub	Holds the user's write 10 output
@INPUT.sub	Holds the user's /input file
@date.SCHED.	Used by TODO to hold user's monthly schedule.
@INI.xxx	Used by FSI

@INI.LISTING	Used to hold job output run under FSI
@INI.sub	Used to hold the FSI options.
@LEARN	Holds the restart info of the LEARN command of IIPS
@LIB	Used by /library command to hold list of files
@MEMO	Used by the now obsolete MEMO facility
@NAMES	Used to hold the MAIL directory.
@PRT	Used by the print screen utility (F9)
@REMIND	Used by REMIND
@REXX.sub	Work file used by REXX
@EXEC.sub	Work file used by the exec command of REXX.
@SENDFILE	Used by the SENDFILE program.
@SORT.TMP	Used by SORT macro for editor.

Qualification

The 17-character file name can have the period characters anywhere *within it* (ie not at the beginning or end). Examples of valid names are RM.LETTER.05DEC77, PROG.V1.L1, THISISALONGNAME.

For example, MAIN.SRC could contain the source and MAIN.OBJ could contain the object deck for the program MAIN.

The qualification can be used to give the appearance of a PDS as in the example: SCRIPT.MAIN, SCRIPT.FORMAT, SCRIPT.SAMPLE, etc.

Notice that CMS's naming conventions of filename/filetype can be directly mapped to filename.filetype.

Specifying Ownership Ids

The ownership id is the userid excluding any subcode. It can prefix the file name as in the example: OWNERID:FILENAME. Note the use of the colon (:) to separate the ownership id from the file name as well as specifying the presence of an ownership id.

If the ownership id starts with an asterisk (*), it takes on the following meanings:

*com:filename	means look into the common library only
*usr:filename	means look into the user's private library only

Temporary Files

Temporary files exist only for the duration of a single job. The only data set name that specifies a temporary file is &&TEMP.

Temporary files are always deleted when closed unless a rename to some permanent name is done at that time.

Internally the system will use a filename of the form .nnnnnn where nnnnn is a number that starts at 1 at IPL time and is increased by one each time a temporary file is allocated. The system will purge any temporary files that exist at IPL time through a routine in the transient phase $PUS.

Save Library Usage Notes

- When one user has a file open for writing, others are locked out as with the UDS system. DISP=WSHR can be used for valid multiple write situations.
- Write Rule. Only affects direct access operation. The system remembers the highest block number written. You may never write beyond this point unless it is the next highest record. This scheme prevents a security exposure that would otherwise result.
- Only one end-of-file (EOF) mark is permitted per file. The UDS system allows any number of end of file marks. There is no special character string that cannot be written to the file. The location of the eof mark is stored in the file's header and is thus not part of the file's data.
- Tape I/O will normally be buffered by the system to correspond to the LRECL and BLKSIZE given on the /FILE command. If RECFM=U is given, then each IO request reads and writes a complete block.
- The minimum record length for reads from file types RDR and TERM is 80 bytes. For other file types the minimum is one byte for RECFM=F and FC files and zero bytes for RECFM=V and VC files.
- Should the file run out of space during a write operation, the system will automatically add secondary extents. The amount of secondary space added as well as the maximum size is controlled by attributes stored with the file.

Assembler Language Interface

Overview

This section describes the macros and control blocks that constitute the assembler language interface to the MUSIC file system. A high level language subroutine interface is also available. It is described in Chapter 22.

In order to use a file the user must OPEN the file. If the file does not exist, there is an option that will have the open create one at this time. Several other options exist on the open and are discussed below. Once the file has been successfully opened, the user can then read and write records to and from the file. Two general techniques exist to read and write to the file. One is the unbuffered technique. With this technique, the user must provide the block number that starts the read or the write request. Logical record deblocking with this form of request is the user's responsibility.

The other, more common access method, is to allow the system to do the logical record deblocking. With this technique, the user need not be concerned with how the logical records are mapped into the physical disk records. There are four record formats: F, FC, V and VC. The meanings of these four formats were discussed in a previous section.

Once the read and write operations have been completed, you then CLOSE the file. On the close operation you can specify that the file be deleted from the system, or kept, or renamed to some other name.

All the requests reference a MFIO argument list. The user may choose to use the same argument list for all files that will be referenced in a program, or the user can choose to have different ones for each file. The argument list is composed of two sections: one a fixed section which must always be given. This is followed by a list of optional argument pointers. Pointers can be given to optional arguments which have no meaning for the specific request currently being used.

WARNING: Do not attempt to use the MFIO interface from within the MUSIC resident system modules. They are designed to be used by code in the User Program or Link Pack areas.

Macros

The following five macros are used. These not only give access to the Save Library file system, but are used to access tapes, UDS files, terminals, and other I/O devices.

MFARG	This macro is used to set up an argument list. Several MFARG's can be given to set up one argument list. The argument list is not generated until a MFGEN macro is used. It is quite common to have several MFARG macros in sequence terminated with a MFGEN macro.
MFGEN	This macro completes the argument list definition defined by a series of MFARG macros.
MFVAR	This macro can be used to create any of the argument blocks pointed to by the MFARG macro.
MFSET	This macro alters the request options in an argument list. It performs no system request but just sets the required bits for a future request. *Register 1 is altered by this macro*.
MFREQ	This macro issues the SVC that performs the request defined in the argument list. *Register 1 is altered by this macro*.

req-name parameter

The *req-name* parameter is used in several macros to specify the type of request. The valid req-name parameters are given below:

OPEN	The open function is to be performed.
CLOSE	The close operation is to be performed.
DIRSRV	A directory service function like CD, MD, or RD is to be performed.
FSIO	Perform 3270 full screen operations.
IO	Perform a read or write or control request using the logical record deblocking routines. The file's RECFM determines how the deblocking will take place.
EXTRACT	Extract information about this file but do not open it. The user must have enough privileges to at least read the file in order that the extract operation be done.
JOBI	Perform the job initialization functions. This call is used internally and can never be done from a user program.
JOBT	Perform the job termination function. This call is used internally and can never be done from a user program.
LIBR	Return the parameters which enable the caller to perform a LIBRARY operation.
MISC	Performs miscellaneous functions.
MSG	Get error message text (treated like an I/O read).

UIO Perform an unbuffered read-write request. The read/write request always starts at the beginning of the 512-byte disk block and continues for as many blocks as required to satisfy the length field. The file's RECFM has no effect on this type of I/O.

USERCTL Read or write a user control record (UCR).

MFARG Macro

The following are the operands that may be placed on this macro:

req-name Specifies the request code to be assembled into the argument list. The request code of numeric 0 may be used if none is to be assembled into the list.

(option1,option2,...)
Specifies the options to be assembled into the argument list. Options are separated from each other by commas and the entire list must be enclosed in parentheses. Refer to the comments under the various request types for valid option names.

optional parameters
Specify the optional parameter name in keyword format followed by the Assembler label of the optional argument. Refer to the separate list below for the valid names.

ULAB= Specifies the Assembler label that will be generated on the one byte internal unit number in the argument list.

U=n Specifies that a unit number of n is to be assembled into the argument list.

RLAB= Specifies the Assembler label to be generated on the one byte return code returned by the various requests.

PICT=Y This parameter can be used to have the macro generate a diagram showing the layout of the argument.

The label field of this macro will generate a corresponding label if it is used on the first MFARG macro in a sequence. Otherwise the label will be associated with the optional parameter pointer given on the macro. Because of this second usage, the user should use separate MFARG macros when the user wishes the labels to be associated with the specific optional parameter names.

The MFARG instructions (one or more) are followed by an MFGEN macro instruction, which causes the argument list to be generated. An optional operand *AREA=location* on MFGEN causes MFGEN to generate executable instructions to create an MFIO argument list at the specified location, using information from the MFARG instructions. This makes it easier for reentrant code to set up an MFIO argument list in a work area defined by a DSECT. A label may be used on MFGEN if the AREA option is present. When AREA is used, MFGEN modifies register 1.

Optional Parameters

The optional parameters on the MFARG macro are given in a keyword format. For example, XNAME=FNAME would be used to specify that the optional parameter XNAME should point to a label called FNAME in the user's program. The valid optional parameter names are given below:

ARG Points to the buffer location and length associated with the current read or write operation. For certain types of I/O, this control block will also contain the starting record number or RBA locator.

BUPNUM	Points to the argument that contains the backup tape number associated with the last backup operation done for this file. This number may be zero indicating no backup done yet for this file. This number can be set by privileged users via the SETBFG option on the CLOSE request.
EOFPT	Points to the argument area which contains the highest block number written and the displacement of the last written byte within that block.
EXTNTS	Points to the argument area which contains the extent list of the file.
FSARG	Points to the 3270 full screen I/O argument. This contains the buffer addresses, lengths and control options used by the interface.
INFIN	Points to the information to be assigned to a new file. Such information includes its records size, its record format, and its access control.
INFOUT	Points to the location of the control block which is to be filled in by the system to contain information about an existing file. INFOUT may point to the same location as INFIN.
LIBR	Points to the optional argument area used in conjunction with the LIBR request.
XNAME	Points to the file name or the ddname of the file (64 characters).
NAME	Points to the file name or the ddname of the file (22 characters). The XNAME parameter should be used for new code as it can handle longer names.
RNAME	Points to the returned file name (64 characters).
PHYS	Points to the control block that will contain the current record length and RBA information after a read or write operation.
TAG	Points to the tag field to be associated with a new file or the location where the system is to move in the tag information of the existing file. The tag information is only returned if the user is the owner of the file.
UCTL	Points to the argument area used to access and modify the user control record for a specific library code. A non-privileged user can only read the user's own UCR record.
HINFO	Points to the argument area that contains the usage information from this file's header. This argument replaces UINFO used before ownership ids could be longer than 4 characters. The only case to use UNIFO is for EXTRACT requests that use the header location on input.
UINFO	This argument was used before ownership ids could be longer than 4 characters. The only case to use UNIFO is for EXTRACT requests that use the header location on input. Use HINFO for other cases.
XINFO	Points to the 40-byte argument area that contains additional usage information from the file's header. The first 4-byte word contains the time of day (in units of 1/300 sec) of creating or last open for write.

MFVAR Macro

This macro is used to generate any optional argument that may be required. The argument name that is to be generated is given as the first operand. The name is the same as the req-name given on the MFARG macro. A keyword parameter of PRE=prefix, may be used to specify a four character prefix to be placed in front of

the generated labels within the argument. The keyword parameter of PICT=Y, can be used to generate a diagram to show the layout of the various fields in this argument.

MFSET Macro

The operands in this macro are as follows:

arg-name This specifies the Assembler label of the argument list to be used for the request. The label is the one given of a MFARG macro.

req-name This specifies the request type to be moved into the argument. Omit this parameter if the request type is not to be changed from a value previously set in the argument.

R=(option1,option2,...)
This is used to specify the request options to replace those in the specified argument.

M=(option1,option2,...)
This specifies the request options that are to be modified in the specified argument. Notice that this form only modifies the specified option bits and keeps the remaining ones untouched. You can omit the req-name operand, meaning that you do not wish to alter the request code in the argument list. You may use a minus sign immediately in front of any option to specify that option is to be turned off in the current argument list.

MFREQ Macro

The options for this macro are as follows:

arg-name Points to the Assembler label associated with an argument list generated by a MFARG macro.

EOF=eofadr Specifies the Assembler label to branch to should an end-of-file condition be found during a read operation.

BAD=badadr Specifies the Assembler label to branch to if an unusual condition occurred during the processing of this request.

OPEN Request

The OPEN request passes a data set name, a ddname, or a PDS member name to the SL and upon the successful completion, the system will set the one byte internal unit number in the argument list.

An OPEN by data set name (dsname) is independent of any /FILE JCL definitions. It is a true dynamic OPEN. Various options are available at open time to say whether an existing file with that name is acceptable and whether a new file is to be created if none exists with that name.

An OPEN by ddname requires that a valid /FILE JCL definition be in effect for this ddname. This OPEN retrieves the internal unit number associated with the ddname. The EOFB field in the EOFPT argument is returned for SL and UDS files. The RSIZ and RFM fields of INFOUT are also returned. For files, the PRM field of INFOUT returns the current file size. The file has already been opened when the /FILE JCL was checked. Therefore, the OKOLD option must always be given. The LU option can be given to mean that the first word of the ddname field is the binary logical unit number.

An OPEN by PDS member name uses the root name given on a valid /FILE JCL definition to form a

dsname. Once formed, the OPEN proceeds similarly to an OPEN by dsname. If the dsname cannot be found, it will create a new file if OKNEW is given. If OKNEW is not given, other root names will be tried in the order specified on the /FILE JCL.

At least one of the options RDOK, WROK must be given.

The various options that can be specified with the OPEN request are as follows:

DDNAME	The name provided is a ddname. Only the ddname OPEN will be attempted.
DDORDS	An attempt will be made to perform an OPEN by ddname. If the ddname is not found, an OPEN by dsname will be done.
MEMBER	An OPEN by PDS name will be done. The name field contains the ddname as the first eight characters followed by the member name as the next eight characters.
LU	Used with DDNAME. The OPEN is to be done on a logical unit number. The full word logical unit number is in the first 4 bytes of the name argument.
OKNEW	Specifies that the OPEN will create a new file if one does not exist.
OKOLD	Specifies that an existing file can satisfy the OPEN request.
RDOK	Specifies that the user will be reading from the file.
WROK	Specifies that the user will be writing to the file.
APPOK	Specifies that the user wishes to write records only to the end of the file. The file will be positioned to the end in anticipation of these writes. The WROK option must also be given.
NODATE	Specifies that the date last opened for reading will not be updated. This option is only valid for privileged users. When the file is opened for writing, this option has no effect.
SETUI	Specifies that the usage information given with the open request is to be used with the new file. This option requires that the user be privileged. This option can use the HINFO or UINFO parameters. Do not use both.
POSEND	Specifies that the file pointers are to be set so that writes done will add records to the file.
NOEFPUP	Do not update the EOF pointer at CLOSE or TCLOSE time. This option is ignored if the RLSE option is used.
NOEFPLO	Do not lower the EOF pointer at CLOSE or TCLOSE time.
ENQSHR	Forces the enqueue operation to be done in share mode. Normally the enqueue operation will be done in share mode only when WROK is not used.
ENQEXCL	Forces the enqueue to be done in exclusive mode. This means that no other user can access the file while this one is open. A user who is not allowed write access to the file cannot use this option. Normally the enqueue operation will be done in exclusive mode when WROK is used.
XONLY	Used only from the module SIOCS to inform MFIO that reads of execute only files are valid at this time.
JOBOPN	Specifies that the file is to be kept open until the end of the job. CLOSE requests on the file

will be handled as if the TCLOSE option was also given. This option is used by the module CTL when opening a file specified on a /FILE statement.

EOJDEL This option is used in conjunction with JOBOPN to delete a file at the end of a job.

EOJREL This option is used in conjunction with JOBOPN to perform the RLSE option at the end of the job.

PGMP Use merged program and user privileges for file access.

DIRNOK Allows the user to create and read the directory file. It also allows the creation of files into a non-existent directory. This option requires the MAINT privilege, otherwise this option is ignored.

NORAM Do not use RAM disk copy of the file. Valid with OKOLD.

CLOSE Request

The following options are defined for the CLOSE operation:

DEL Specifies that the file is to be deleted from the library.

REPL Specifies that a new file name is to be associated with the existing file. If a file already exists with that name, it is to be deleted. If a file does not exist, then this option will have the same effect as the RENAME option. New access control flags are assigned to the new name from the INFIN argument that should be provided when this option is used. Other parts of the INFIN and TAG arguments are not used by CLOSE. The new name cannot be the same as the original file name. This option does not set the RNAME field.

RENAME Specifies that the file is to be renamed to another name only if that new name does not exist on the library. New access control flags are assigned to the new name from the INFIN argument that should be provided when this option is used. Other parts of the INFIN and TAG arguments are not used by CLOSE, except for logical record length when the SETEFP option is used. This option does not set the RNAME field.

RLSE Specifies that any unused space at the end of the file is to be released. This means that the file is to be made as small as possible to hold the existing data.

TCLOSE Specifies that the CLOSE function is to be performed but that the file is to remain open for further requests. The current end of data pointer is updated in the file's header. RLSE is the only other option that can be given when TCLOSE is used.

SETBFG Specifies that the backed up bit in the index entries for this file are to be set. The backup tape number specified by the BUPNUM is also to be set in the index. This option is only valid for privileged users.

SETEFP Specifies that the pointer to the end of the written data is to be updated with the value specified in the EOFPT argument. The file's record size will be set to the value given in the RSIZ field of the INFIN argument. This allows the maximum record size to be set for a variable length file. The SETEFP option is only valid for privileged users and exists primarily for the MFREST utility and the COPY command.

GETEFP Refresh the EOF pointer. The file remains open. This option does not decrease the EOF pointer. This option cannot be used with other CLOSE options.

RESET Do not do pending I/O, but update header block, setting line count to zero and end-of-file pointer to the start of the file. This sequence is logically equivalent to, but faster than, a rewind operation followed by a write end-of-file. The file is closed.

CTAG Set a new tag field when the file is closed. The new tag is taken from the TAG argument.

PGMP Use merged program and user privileges for file access. This option is used with REPL or RENAME.

DIRNOK Allows the user to create and read the directory file. It also allows the creation of files into a non-existent directory. This option requires the MAINT privilege, otherwise this option is ignored.

The above options can be used in combination. Should a request such as RENAME fail to work, the user may issue additional CLOSE requests. When the CLOSE requests is successful, the unit number in the argument list will be set to 0.

IO and UIO Requests

These requests are used to perform read, write and control operations on the file. The valid options are as follows:

LU The unit number in the argument list is a logical unit 1 to 15. The system will use this number to index into $LUXTAB to determine the internal unit to use. This option is used mainly by FORTRAN to perform its I/O. ($LUXTAB is initialized by CTL and may be altered by giving /FILE JCL with numeric ddname.)

RD Read a record from the file.

WR Write a record to the file.

WEOF Specifies that an end-of-file operation is to be done on the file. Not valid with the UIO request. (Notice that with the SL, only one end-of-file marker can exist on the file.)

REW Specifies that a rewind operation is to be done on the file. Not valid with the UIO request.

BSP Specifies that a backspace operation is to be done on the file. Not valid with UIO request and SL files.

FILL Specifies that the user's buffer is to be filled with blanks should the current record read from the file be smaller than the user's buffer. Not specifying this option is no guarantee that the fill function will not be done. Not valid with the UIO request.

TRUNC This specifies that blank characters are to be truncated from the record that is being written to the file. Only meaningful for RECFM=V or VC files. The record will then appear to have no trailing blanks on it. Not valid with the UIO request.

RBA The IO request is to start at the specific location within the file given by the relative byte address (RBA) contained in the ARG parameter. Use this operation with care if write operations are being performed. This option is not valid with UIO requests.

USERL The caller's request length is to override the file's record size for the current IO operation of RECFM=F files. This option is used by FORTRAN when doing direct access operations.

EXTRACT Request

This request is used to obtain information about a file without opening it. (An OPEN request can also obtain the same information.) The options are as follows:

DIRLOC Specifies that the location of the file's header is given in the UINFO argument. Do not use the HINFO parameter. (Note the "header" used to be called the file's "directory". The name has been changed to avoid confusion with directories that the user see.) In this case the UINFO argument is 4 bytes in the form 00XXYYYY where XX is the library number and YYYY is the space unit number (as in the index entry). This saves the index search that would otherwise be done. This option can only be used from a privileged program. It is used by the LIBRARY command. Note that the file name argument must still be provided when DIRLOC is used.

PGMP Use merged program and user privileges for file access.

DIRSRV Request

The Directory Service request usually takes a directory name as input. Upon the successful completion, the system will return a 64 byte RNAME field as a result.

The directory name can either be passed to the DIRSRV request in the NAME field if the directory is 22 characters or less, or in the XNAME field if the directory is 64 characters or less.

Directory names prefixed by with strings like "..\", ".\", and ".." will be resolved by using the current directory.

The various options that can be specified with the DIRSRV request are as follows:

DIRCD Changes directories from the current directory to the specified directory.

DIRMD Creates a new directory.

DIRRD Removes an empty directory.

DIRQD Queries the current directory. Works like DIRCD but does not change the current directory.

MSG Request

This request is used to obtain the message text corresponding to the error code set in the argument list. Should not be called if error code is 0. The ARG argument defines the location and maximum length for the returned message. A length of 70 is considered ample for message texts. Unused location in the message buffer will be blanked.

USERCTL Request

This request is used to read and write a UCR record. A non-privileged user can only read the user's own UCR record. The options are as follows:

SETUCR Specifies that the maximum allocation limit fields in the UCR are to be replaced with the ones specified in the calling argument.

REPUCR Specifies that the user control record is to be replaced entirely with the one specified in the

calling argument.

CODE Specifies that the ownership id associated with the UCR record is to be taken from the NAME argument. Otherwise the ownership id will be that of the current user. Maximum length is 16 characters with a blank terminating the field if shorter.

DELUCR Specifies that the UCR record for this ownership id is to be deleted from the index.

LIBR Request

This request returns arguments to enable the caller to do a scan of the Save Library Index. For example, the LIBRARY command issues this request to determine the first and last block numbers of the index it will have to read. If the caller passes a 4-character ownership code, then only the block numbers corresponding to that segment will be returned.

LIBNAM Gets name from XNAME field instead of from the LIBR parameter.

MISC Request

This request is used to perform miscellaneous functions as follows:

SETLUX Takes the one byte internal unit number given in the basic argument and places it in the logical unit cross reference table (LUXTAB) at location corresponding to the full word logical unit number given in the NAME argument.

SETDDN Adds a ddname to the ddname table (if not already there), and associates an internal unit number with the ddname. The step number in the table entry is set to zero, so that the ddname will apply to all steps of the job. The ddname is taken from the first 8 bytes of the file name argument. The internal unit number is taken from the basic argument block. If there is not enough room in the ddname table, the ddname is not added, and error 31 is returned.

FSIO Request

This request is used to perform full screen I/O operations to 3270 type terminals. The basic request block should contain pointers to the FSARG and PHYS blocks. U=9 should also be specified. The FSARG block contains lengths and pointers to the 3270 data stream buffers. The actual length of the input read is returned in the PHYS block. *Chapter 22 - System Programming* contains detailed information of the assembler and subroutine interfaces to this facility.

File System Messages and Return Codes

Special Conditions

```
Error  1  END OF DATA SET ENCOUNTERED (see also error 44)
Error  2  INCORRECT LENGTH
```

Errors in Calling ARG

```
Error 10  INVALID REQ
```

```
Error 11  INVALID REQ PARAMETER
Error 12  FILE NAME INVALID
Error 19  INVALID ARGUMENTS IN CALL TO SERVICE SUBROUTINE
```

Violations of MUSIC Conventions

```
Error 20   TOO MANY OPEN FILES
Error 21   NOT YOUR LIBRARY
          (Not authorized to store under this code.)
Error 22   NOT YOUR FILE
Error 23   VIOLATION OF WRITE RULE
Error 24   ATTEMPT TO READ BEYOND END OF WRITTEN INFO
Error 25   WRITE THEN READ SEQ INVALID
Error 26   YOUR USERID CANNOT CREATE FILES ACCESSIBLE BY OTHERS
Error 27   YOUR USERID CANNOT CREATE FILES IN THE COMMON INDEX
```

File Existence Errors

```
Error 30   FILE NOT FOUND
Error 31   DDNAME NOT FOUND (see also error 35)
           (on a SETDDN request, error 31 means that there
           is not enough room in the ddname table)
Error 32   FILE ALREADY EXISTS
Error 33   FILE IN USE
Error 34   COMMON NAME USED BY SOMEONE ELSE
Error 35   UNIT NUMBER NOT DEFINED
           (or ddname undefined by user request such as by
           specifying UNDEF on a /FILE statement.)
Error 36   SUBDIRECTORY DOES NOT EXIST
```

Restrictions Imposed by FILE

```
Error 40   SPACE QUOTA EXCEEDED FOR THIS USERID
Error 41   SPACE QUOTA EXCEEDED FOR THIS FILE
Error 42   CANNOT ADD SPACE TO THIS FILE
Error 43   REQUESTED ACCESS OR OPERATION NOT ALLOWED
Error 44   REQ BEYOND EXTENT OF FILE
Error 45   FILE RECFM NOT DEFINED
Error 46   FILE CANNOT BE READ SEQUENTIALLY
Error 47   INSUFFICIENT SPACE FOR BUFFER ALLOC
Error 48   MIN RECORD LEN IS 80 FOR THIS FILE TYPE
```

System Temporary Limits

```
Error 50   FILE NOT ON-LINE
Error 51   NOT ENOUGH FREE DISK SPACE
Error 52   NOT ENOUGH FREE DISK SPACE (IDX)
```

System Integrity Errors

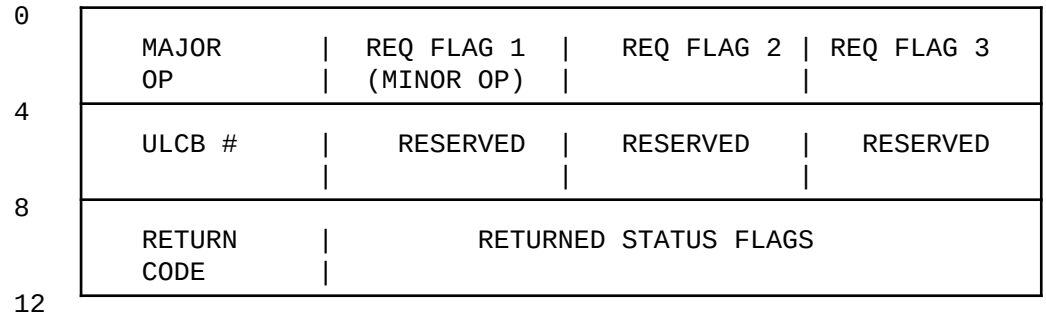
```
Error 60   RD I/O ERROR IN FILE
Error 61   WR I/O ERROR IN FILE
Error 62   RD I/O ERROR IN SYSTEM AREA
Error 63   WR I/O ERROR IN SYSTEM AREA
Error 64   INDEX IN ERROR
ERROR 65   HEADER IN ERROR
Error 66   MAP INTEGRITY ERROR
Error 67   INDEX/HEADER MISMATCH
```

System Coding Errors

```
Error 70   SYSTEM FILE ERROR (logic error in system module)
```

Control Blocks

Basic Caller's Request Block



NAME--Data Set or DDNAME (old format 22 character name)

```
    Same as XNAME but is only 22 characters long

          Label     DC    CL22' '            NAME
```

XNAME---Data Set or DDNAME

```
    CAN BE IN 4 FORMS:

       1) DSNAME   EG. CODE:FILENAMEXXXXXXX (MAX 64 CHARS)
                   EG. FILENAMEXXXXXXX (MAX 50 CHARS)
                   EG. DIR1\DIR2\DIR3\FILENAME (MAX 50 CHARS)
       2) DDNAME   EG. DDDDDDDD (MAX 8 CHARS)
       3) DDNAME AND MEMBER NAME EG. DDDDDDDDMMMMMMMM (MAX 16 CHARS)
                                 EG. DDDD    MMMMMM   (MAX 16 CHARS)
       4) BINARY LOGICAL UNIT NUMBER (4 BYTES)

          Label   DC    CL64' '            XNAME
```

RNAME---Returned NAME from MFIO

RNAME is a 64 byte long field

General format is

```
CODE:DIR1\DIR2\FILENAME
```

One use for this arg is to return the filename actually found on an open request.

```
Label   DC    CL64' '            RNAME
```

INFIN/INFOUT---Size and Attributes

```
0
      +-------------------------------------------------------+
      | INFIN: PRIMARY SPACE ALLOC IN K BYTES                 |
      | INFOUT: CURRENT FILE SIZE IN K BYTES                  |
4     +-------------------------------------------------------+
      | SECONDARY SPACE ALLOC.  IF>0 THEN IS AMT IN K BYTES   |
      |                         IF=0 THEN AMT IS CURRENT + 50%|
      |                         IF-N THEN AMT IS CURRENT + N% |
8     +-------------------------------------------------------+
      | MAX SPACE LIMIT FOR FILE IN K BYTES                   |
12    +-------------------------------------------------------+
      |  INFIN: RSIZ IF RECFM=F,FC |   RECFM      |  (RESERVED)|
      |  INFOUT: MAX RSIZ WRITTEN  |              |            |
16    +-------------------------------------------------------+
      | <----------------FILE ACCESS CONTROL FLAGS------------->|
      | GEN CTL INFO| OWNER ACCESS  | NOBODIES ACC | RESERVED |
20    +-------------------------------------------------------+
```

```
Label     DS    0F
 **FILE SIZE SPECIFICATIONS
xxxxPRM   DC    A(0)             PRIMARY SPACE WANTED IN K BYTES
xxxxSEC   DC    A(0)             IF >0 THEN IT IS AMT IN K BYTES
                                 IF =0 THEN AMT IS CURRENT + 50 %
                                 IF -N THEN AMT IS CURRENT + N %
xxxxMAX   DC    A(0)             SPACE LIMIT IN K BYTES
                                 -1 MEANS UNLIMITED

 **LOGICAL RECORD CONTROL
xxxxRSIZ  DC    AL2(0)           RSIZ IF RECFM=F OR FC
xxxxRFM   DC    AL1(0)           RECFM
                                 =X'00' RECFM=U (NO LOGICAL RECFM DEF)
                                 =X'01' RECFM=F (FIXED)
                                 =X'02' RECFM=FC (FIXED COMPRESSED)
                                 =X'03' RECFM=V (VARIABLE)
                                 =X'04' RECFM=VC (VAR COMPRESSED)
          DC    AL1(0)           RESERVED FOR CARR CTL TYPE
```

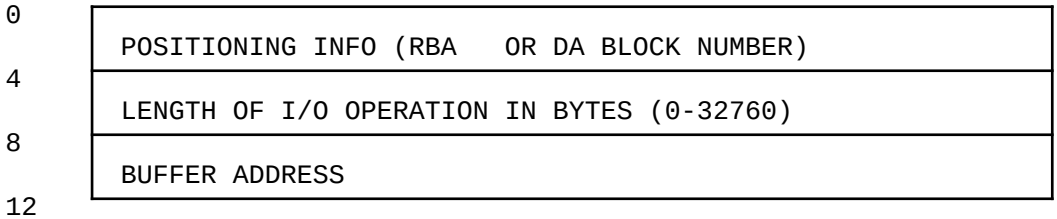
GENERAL CONTROL FLAGS

```
          EQU   B'10000000'      FILE IS IN COMM INDEX
          EQU   B'01000000'      MAINTAIN USAGE COUNTS IN DIRECT
          EQU   B'00100000'      FILE IS A VSAM FILE

xxxxGCTL  DC    X'00'            GENERAL CONTROL FLAGS
```

ACCESS CONTROL BITS (USED FOR NEXT 3 BYTES)

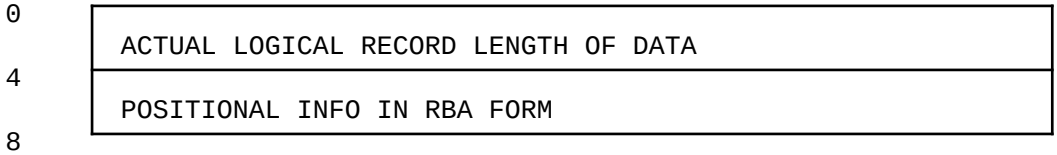
```
      NOTES:
         -BITS NORD+NOWR MEANS NO ACCESS
         -IF XO TO OWNER, NO ATTRIBUTES CAN BE CHANGED
         -ONLY THE OWNER (OR A PRIVILEGED USER) CAN REPLACE OR
          DELETE THE FILE

 ****

          EQU   B'10000000'        NO KIND OF READ ACCESS ALLOWED
          EQU   B'01000000'        NO KIND OF WRITE ACCESS ALLOWED
          EQU   B'00100000'        ONLY RD ACCESS IS EXEC-ONLY
          EQU   B'00010000'        ONLY WR ACCESS IS APPEND

xxxxACOW  DC    X'00'              ACCESS CONTROL FOR OWNERS
xxxxACNB  DC    X'00'              ACCESS CONTROL FOR NOBODIES
xxxxACPJ  DC    X'00'              ACCESS CONTROL FOR PROJECT MEMBERS
```

ARG---Read/Write Arg



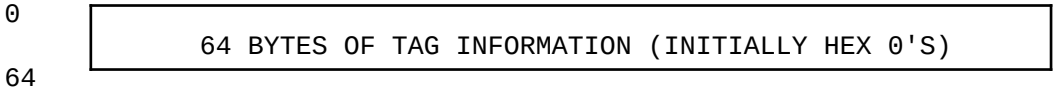

```
Label     DS    0F
 **THE FOLLOWING WORD IS USED TO PASS POSITIONING INFO
xxxxSBNU  DS    0F                 STARTING BLK NUMBER (UIO ONLY)
xxxxIRBA  DS    0F                 RBA FOR RWB REQUESTS ONLY
          DC    A(0)               POSITIONING INFO (SEE ABOVE)

xxxxRLEN  DC    A(0)               LENGTH OF REQ (BYTES)
xxxxRBUF  DC    A(0)               LOCATION OF BUFFER
```

PHYS---Returned Physical Info



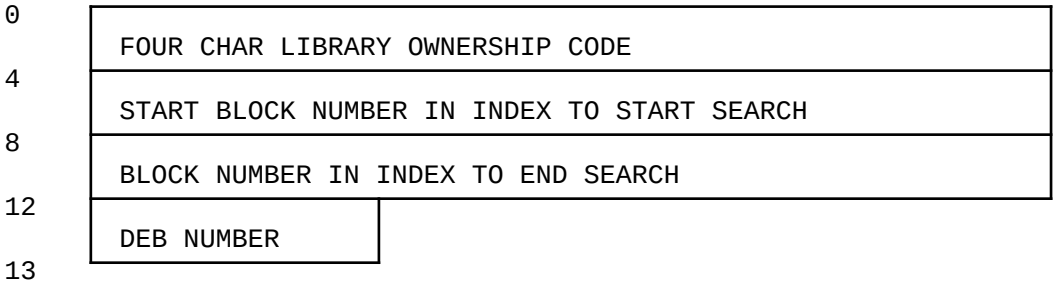
```
Label     DS    0F
 *THESE ARGS ARE VALID ONLY FOR BUFFERED READ/WRITE REQUESTS
xxxxARSZ  DC    A(0)               ACTUAL LOGICAL RCD LEN ON DISK
                                  (IE LENGTH OF VAR RECORD BUT NOT
                                  NUM OF BYTES IN COMPRESSED RCD)
xxxxORBA  DC    A(0)               RBA OF START OF THIS RECORD
```

TAG---Tag Information



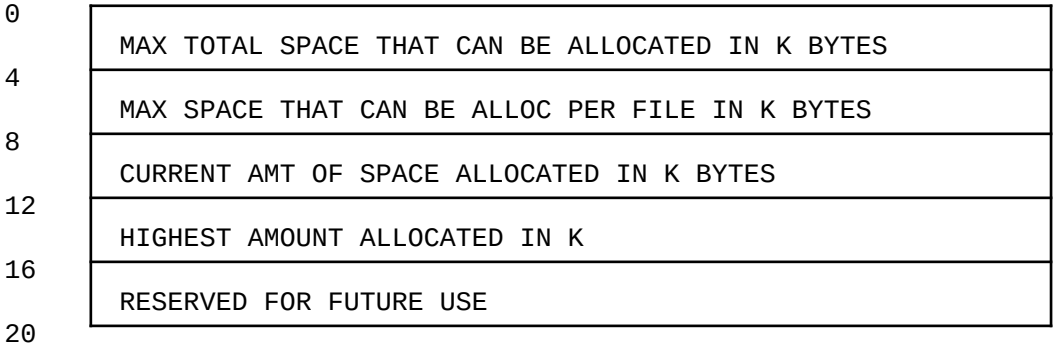

```
Label     DC    XL64'00'           TAG FIELD
```

LIBR---Arg for LIBRARY Req



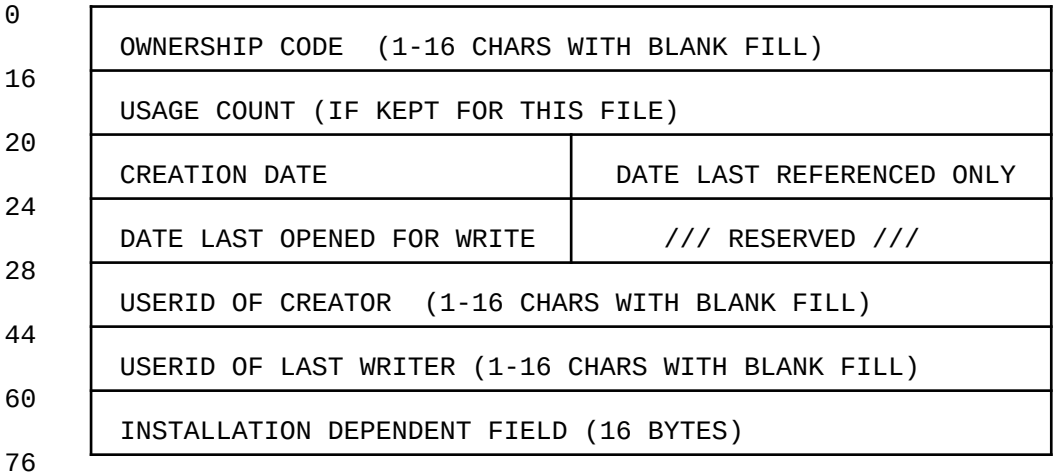

```
Label     DS    0F
xxxxLBCD  DC    CL4'   '           OWNERSHIP ID FOR LIBR OPERATION
xxxxLBSB  DC    A(0)               START BLOCK NUMBER OF SEGMENT
xxxxLBEB  DC    A(0)               ENDING BLOCK NUMBER OF SEGMENT
xxxxLBDB  DC    AL1(0)             DEB NUM OF INDEX DATA SET
```

UCTL---User Control Record Arg



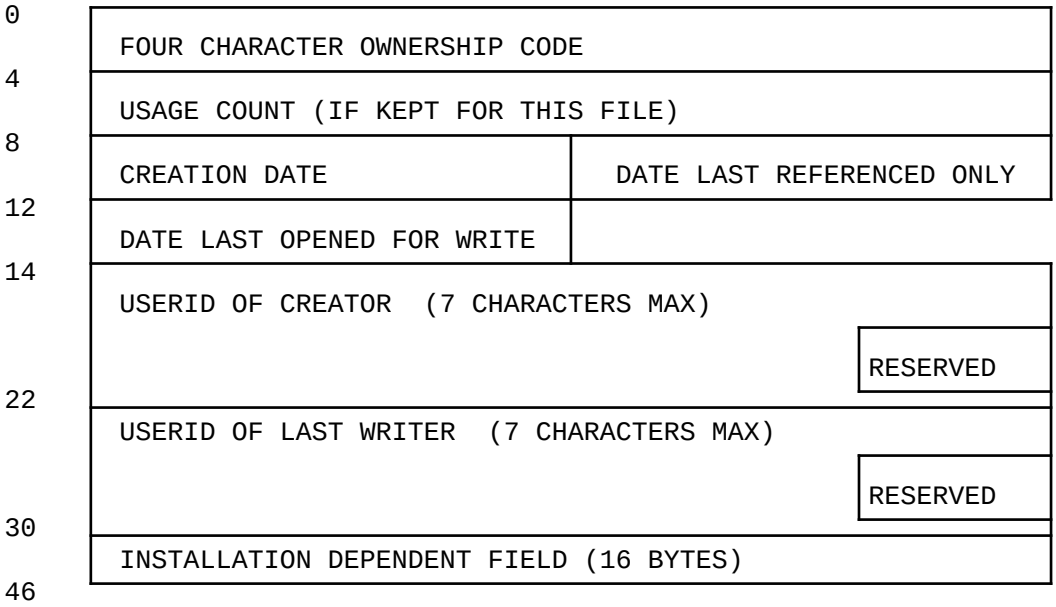
```
Label     DS    0F                **ALL UNITS ARE K BYTES (K=1024)
xxxxMAXS  DC    A(0)              MAX SPACE THAN CAN BE ALLOC TOTAL
xxxxMAXF  DC    A(0)              MAX SPACE PER FILE

xxxxACUR  DC    A(0)              AMT CURRENT ALLOCATED
xxxxAHWM  DC    A(0)              HIGHEST AMOUNT ALLOCATED
          DC    A(0)              RESERVED FOR FUTURE USE
```

HINFO--Usage Information from File's Header for 16 char userids



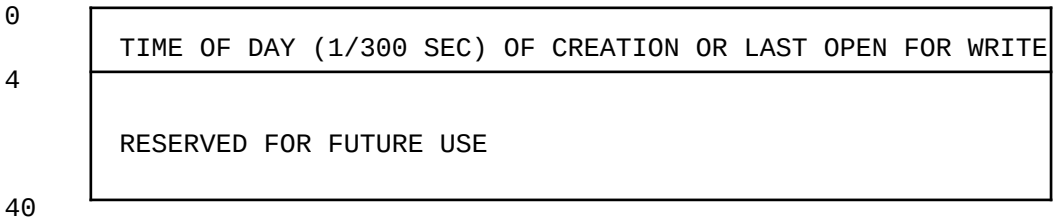

```
 *NOTE
 *This argument replaces UINFO used before ownership ids
 *could be longer than 4 characters.
 *The only case to use UNIFO is for EXTRACT requests that
 *use the header location on input.

 Label    DS    0F
 *NOTE:THE OWNERSHIP AND USERIDS ARE 1 TO 16 BYTES
 *       LONG WITH BLANK FILL.
 xxxxHIFC DC    CL16' '           OWNERSHIP ID OF CREATOR
 xxxxHIUC DC    F'0'              FILE USAGE COUNT (IF KEPT)

 **DATE INFORMATION.
 *   DATE NUMBER IS DAYNUM+(YEAR-1970)*366
 xxxxHICD DC    AL2(0)            DATE THIS DIRECTORY ITEM WAS CREATED
 xxxxHIRD DC    AL2(0)            DATE FILE WAS OPENED ONLY FOR READING
 *                                (MAY BE =0)
 xxxxHIMD DC    AL2(0)            DATE FILE WAS OPENED FOR WRITE
          DC    AL2(0)            RESERVED

 xxxxHICI DC    CL16' '           USERID OF CREATOR
 xxxxHIWI DC    CL16' '           USERID OF LAST WRITER

 xxxxHIID DC    XL16'00'          INSTALLATION DEPENDENT FIELD
```

UINFO--Usage Information from File's Header



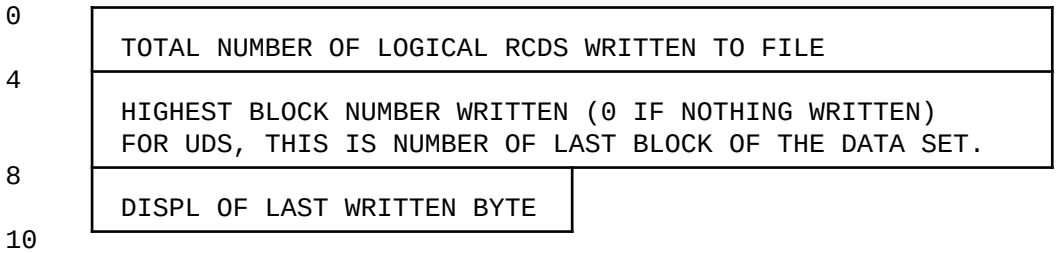

```
Label     DS    0F
 *NOTE THE FIRST WORD IS USED TO PASS THE HEADER LOC ON
 *CERTAIN TYPES OF EXTRACT REQUESTS.
xxxxUIFC  DC    CL4' '           OWNERSHIP ID PART OF FILE NAME.
xxxxUIUC  DC    F'0'             FILE USAGE COUNT (IF KEPT)

 **DATE INFORMATION.
    DATE NUMBER IS DAYNUM+(YEAR-1970)*366
xxxxUICD  DC    AL2(0)           DATE FILE WAS CREATED
xxxxUIRD  DC    AL2(0)           DATE FILE WAS OPENED ONLY FOR READING
                                 (MAY BE 0)
xxxxUIMD  DC    AL2(0)           DATE FILE WAS OPENED FOR WRITE

xxxxUICI  DC    CL7' '           USERID OF FILE'S CREATOR
          DC    C' '             RESERVED
xxxxUIWI  DC    CL7' '           USERID OF LAST WRITER
          DC    C' '             RESERVED

xxxxUIID  DC    XL16'00'         INSTALLATION DEPENDENT FIELD
```

XINFO--Extra Usage Information from File's Header



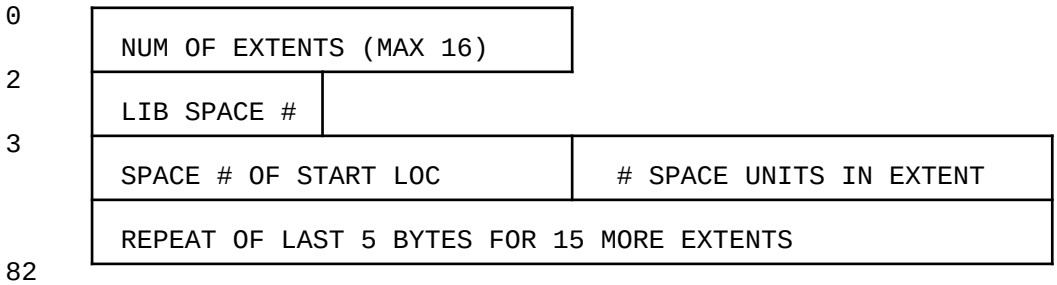
```
Label     DS    0F
xxxxXITD  DC    F'0'          TIME OF DAY (1/300 SEC) OF CREATION
                              OR LAST OPEN FOR WRITE (OR 0 IF BEFORE
                              TOD SUPPORT BEGAN)
xxxxXIRS  DC    9F'0'         RESERVED FOR FUTURE USE
```

EOFPT--Pointers to End of File



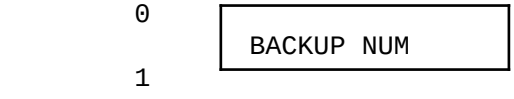

```
Label     DS    0F

      THE FOLLOWING IS THE NUMBER OF LOGICAL RECORDS WRITTEN TO THE
      FILE.
      THIS NUMBER MAY BE 0 EVEN IF THE FILE CONTAINS INFORMATION AS
      THIS FIELD CANNOT BE CORRECTLY MAINTAINED IN ALL CASES.

xxxxNLRC  DC    A(0)

     THE FOLLOWING IS THE 512-BYTE BLOCK NUMBER LAST WRITTEN.
     IF NUMBER IS 0, THEN THE FILE IS EMPTY.

xxxxEOFB  DC    A(0)

     THE FOLLOWING IS DISPLACEMENT INTO THE BLOCK OF THE 1ST BYTE
     BEYOND THAT LAST WRITTEN.
     FOR EXAMPLE, =512 IF BLOCK ALL WRITTEN.
     (CAUTION--THIS NUMBER MAY NOT BE 0 FOR AN EMPTY FILE.)
xxxxEOFD  DC    AL2(0)
```

EXTNTS--Extent List of File



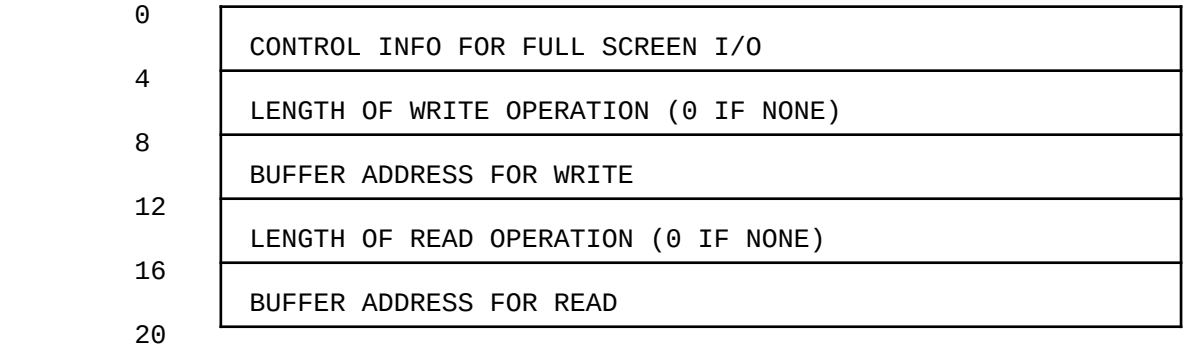
```
Label     DS    0H
xxxxEXTC  DC    AL2(0)           COUNT OF NUMBER OF EXTENTS (MAX 16)

 ** EXTENT INFO FOLLOWS.
       PER EXTENT: FIRST BYTE IS LIB SPACE NUMBER WHERE EXTENT IS
                   LOCATED (1,2,...)
                   NEXT HALF WORD IS STARTING SPACE UNIT NUMBER OF
                   EXTENT (1,2,...)
                   NEXT HALF WORD IS NUMBER OF SPACE UNITS IN EXTENT.
xxxxEXTI  DC    16XL5'00'        EXTENT INFO
```

BUPNUM--Backup Tape Number




```
Label     DC    AL1(0)             BACKUP TAPE NUMBER
```

FSARG--FSIO ARG




```
    Label     DS    0F
    xxxxFSFG  DS    XL4'00'        FLAGS TO CONTROL FULL SCREEN I/O
    xxxxFSWL  DC    A(0)           WRITE LENGTH
    xxxxFSWB  DC    A(0)           WRITE BUFFER LOCATION
    xxxxFSRL  DC    A(0)           READ LENGTH
    xxxxFSRB  DC    A(0)           READ BUFFER LOCATION
```

The control flag definitions are described in the FSIO section of Chapter 22.

Internals - Save Library

Tree-Structured Hierarchical Processing

By default, the current directory is set to the root directory when the user signs on to MUSIC. The character string associated with the current directory is stored in the XTCB control block. It can be set and queried by the DIRSRV service request of MFIO or by the CD command. If a user issues the "CD \ABC\DEF" command, the field in the XTCB will be set to:

```
AL1(8),C'ABC\DEF\'
```

A count of hex 0 would indicate the root directory.

Directory names are stored in the file system like ordinary user files. Their name always ends in a backslash (\) character. Suppose a user creates a directory using the "MD SAMPLE" command. This will cause the file system to create a file called "SAMPLE\". This file is stored like any other file that a user may own. The contents of the file are not used. This is very different to many other systems which actually store the names of all the files that are part of a directory with a directory file. The main use of the directory file on MUSIC is to establish that the user has issued a MD command.

You cannot normally create or delete a directory file. That should be done with the MD or RD commands or by the equivalent DIRSRV MFIO request. The MUSIC backup and restore utilities can directly create and delete directory names. This is done through the use of the DIRNOK special request on some MFIO calls.

Suppose a user creates a file called TEST after he has set the current directory to "SAMPLE". The file will be stored under the name of "SAMPLE\TEST". This allows for direct access to all files on the system without having to read any directory files first.

Data Set Overview

The Save Library consists of a number of data sets. All these data sets have a blocksize of 512 bytes. There is one index data set that contains the names of all the files. The remaining data sets contain the actual data. The Save Library can be expanded by adding more of these data sets, however, there is only one index data set.

Index Data Set

The index data set contains the names of all files and a pointer to the start of each one. It also contains the UCR records. The first few blocks in this data set contain the system control data that is used at IPL time. Such information includes hashing numbers and a field that contains the number of data spaces. The remainder of the index data set is split into several sectors. All the file names belonging to any one given user will be found entirely within one sector. Thus, the LIBRARY command need only search one sector to find the names of all the files stored under one code. There is an overflow sector in the index data set that contains records which could not fit within the proper sector because there was no space in the specific record that was pointed to by the hashing process.

In order to locate a specific file the system first takes the code part of the file name and determines what sector it maps to. It then takes the file name, and using the hashing process, locates the specific disk block within that sector that would contain the specified file name. Upon reading the block, the system can determine whether the file does in fact exist. (The hashing scheme is set so that if multiple users who are sharing the same index sector use the same file name, it will probably map to different index blocks.)

To locate an item in the common library, the system considers all the primary sectors as being one large sector and hashes the file name to locate a specific index block.

Space Data Set

Each space data set starts off with several blocks that contain control information. One of the blocks contains a bitmap which indicates which space units are used within this data set.

The remainder of this data set is used to contain users' data. The first block of any user file is reserved for system use. This block contains the file header. In the file header are the various attributes and

characteristics of the user's file. (The header block used to be called the directory block. It was renamed to avoid confusion with directories that the user can create.)

Bitmap

The bitmap is stored in up to 7 512-byte blocks that start at block 2. The first four bytes contain a check sum. This field is used to check the validity of the map and is calculated by an arithmetic process involving the addition of the remaining words in the map. (Consult the system code in module MFIO for a complete description of how this is calculated.)

The check sum is followed by a two byte bit count containing the number of free space units in this data set.

The bit count field is followed by a bitmap. The bitmap shows exactly which units are free. Each bit corresponds in order with the space unit number available. Thus the bit after the counter is the status of space unit one, the next, space unit two. If the bit is on, then the space unit is free and hence, can be allocated.

File Header

The first block of each file contains its header. This header contains such things as the access control information, the ownership id (userid without subcode), usage dates and the file's extent information.

ULCBs

A set of User Library Control Blocks (ULCB) exist in the service area of the user region. These control blocks contain information about open files. They are contained in the system module MFIOCB.

Buffer Pool

A buffer pool exists for the use of Save Library I/O. This pool is located in the system module MFIOCB. Each buffer in the pool can be assigned to any file. They are assigned dynamically as required using a scheme similar to a paging scheme.

Logical Record Formats

The following describes how the logical records are stored on the Save Library's 512-byte physical blocks.

Fixed Format (F)

With this format, the first logical record contained in any disk block always starts at the first byte of the disk block. When the record size is not greater than 512, then no logical record spans a disk block. Unused space may therefore exist at the end of each block.

When the logical record size is greater than 512, each logical record starts at the beginning of a disk block. Overflow from one block is continued at the first byte of the next block. Unused space may exist at the end of the last block used for each record.

Variable Format (V)

The first two (2) bytes of each disk block are used to contain the displacement to the start of the first logical record in that block. The remaining 510 bytes are always used. Logical records exceeding one disk block will continue at the third byte of the following block. Each logical record is preceded by a 2 byte count containing the number of bytes in the record plus two.

Compressed Format (FC and VC)

The only difference between FC and VC is the meaning of the file's record size. For FC files, the record size means that all records will appear to be this size. Trailing blanks will always be removed from FC files before compression and supplied on expansion. For VC files, the record size means the length of the largest record in the file so far.

The first two (2) bytes of each disk block is used to contain the displacement to the start of the first logical record in that block. If no logical record starts in this block, the first two bytes are zero. The remaining 510 bytes are always used. Logical records exceeding one disk block will continue at the third byte of the following block. Each logical record may be composed of a number of segments. Each starts with a 1 byte descriptor. A repeated text segment is used when four or more bytes in sequence are the same. One byte segments are also represented by the repeated text format. The three types of segments are:

1. *Non-repeated text*. Format: D string
   The D byte is the length of the string minus one. D may range from X'01' to X'7F' representing string lengths of 2 through 128 bytes. D may not be zero.

2. *Repeated text*. Format: D C
   The D byte is the number of times to repeat the character C plus X'7F'. D may range between X'80' and X'FE' representing a repeat count of 1 to 127.

3. *Record end*. Format: X'FF'
   This segment indicates the logical end of the record.

Working with Files

- Reference to another user's private file is done by prefixing the ownership id to the file name. The ownership id is usually the same as the userid. The only exception is when subcodes are used. Thus ABCD:HISFILE refers to the file HISFILE belonging to userid ABCD. This ownership id prefix is accepted in EDIT, RENAME, SAVE, PURGE and other similar commands.

- To change ownership of files you can use the RENAME command. To change a long list of files, use the FILECH utility. The COPYCOL command of the Editor is useful in the preparation of the input to this program.

- Use the TAG command of the Editor to help keep track of the contents of files. This is particularly useful when you own a large number of files.

- A COM option can be specified on the Editor FILE and SAVE commands. This has the effect of forcing the file name to be placed in the common index. A private file in the common index can be accessed only by its owner or by a privileged user. This feature can be used to reserve names in the common index, or to share files between privileged users without the need for public files.

- The "LIBRARY abcd:*" command lists all the files belonging to the ownership id ABCD. The

"LIBRARY ab*:*" command lists all the files belonging to the set of ownership ids that start with the letters AB.

You can select only certain names within a ownership id. The "LIBRARY *:@learn" command lists all the files called @LEARN belonging to anyone.

- Use "LIBRARY *:* COM" to list all files in the common index. These files were saved with the COM or PUBL option.
- Use the ATTRIB utility to find the owner of a public file. Thus "ATTRIB xxxx" displays who owns the file *xxxx.*
- The FPRINT utility is useful to list a large number of files with page headings.
- To purge a number of files, first create a file that contains a list of the file names. Then use the "PURGE <list" command.
- Even privileged users will be unable to store files under unused userids. This is because a UCR record for the userid must exist before files can be stored with that userid. UCR records are automatically added by CODUPD when userids are added. You can use the UCR utility to add UCR records directly without requiring a corresponding sign-on userid.
- The following technique can be used to activate the usage counting of a file. Edit the file and type the editor command FILE * CNT.
- The FIXINDEX and MFHASH utilities are useful for direct inspection and alteration of the Save Library Index should that be required.
- Note that when you set up an execute-only file, you should put in a /SYS NOPRINT statement in the file so that the system will know not to print any /FILE statements that follow.
- You can specify a ownership id on the CD command by using the following format:

```
CD \ownerid:\dir
```

Working with UDS Files

This topic contains tips and techniques for working with User Data Set (UDS) files. These are OS style data sets that are supported by MUSIC. Each file has its own VTOC entry and may occupy from one to five extents. Despite the name, most users' data is stored in the Save Library and UDS files are only kept around for compatibility with past releases. An entry in the user's profile sets the maximum number of tracks that can be used for a UDS file.

- The LISTV command of the DSKDMP utility is used to list the contents of the VTOC. This gives a comprehensive list of all UDS (and SDS) files on any disk. It is also useful in determining the amount of free space on the disk.
- The EDITOR can edit UDS files. Define the UDS file as logical unit 4. Remember to specify the OLD option on the /FILE statement if you want to file the changes.
- When UDS files are deleted, their contents are not erased. Therefore, subsequent users who allocate UDS files may be able to read the information stored there by the previous user of that disk space. (This is not the case with regular files.) The ZERO.FILE utility described in the *MUSIC/SP User's Reference*

Guide, can be used to clear the contents of a file that contains sensitive information.

- To be able to list the UDS files owned by specific users, first create a file SYSDSL which is the same as the file DSLIST with the last line removed. Then run the following job:

```
/INC SYSDSL
VOL='volume1,'volume2',DSNS='userid*'
```

where *volume1*,*volume2* are the names of the UDS volumes to scan and *userid* is the file ownership id of the files that you want to list. The DSNS parameter can specify a data set name pattern, containing wild characters * and ?.

Chapter 21. Load Library and Link Pack Area

Load Library

The system load library is in the dataset SYS1.MUSIC.LOADLIB. Private load libraries can be created in UDS files. These libraries contains relocatable machine language programs that are loaded into memory using the SVC $LODSVC. The OS LOAD and LINK macro instruction can also be used to load programs from these libraries. The system load library contains the modules for system utilities, the editor, the assembler, compilers, system interfaces and commands. Private libraries are used when specific applications require large program libraries. A example of this is the PL/I transient library.

The LDLIBE utility program is used to create and maintain load libraries, LDLIST lists information about the contents of load libraries, and SYSREP can be used to make machine language modifications (ZAPs or REPs) to the programs in a load library.

System performance can be significantly improved by placing programs from the system load library in either the Fixed or Pageable Link Pack Areas. In addition to the loading overhead being reduced, this also reduces system paging and swapping loads.

Modules in the Fixed Link Pack Area (FLPA) are loaded into memory during system initialization. If the module is reentrant it can be executed directly from the FLPA. The pages of the module are never involved in paging or swapping. If the module is not reentrant, it is copied to the user region before execution. This memory to memory load is much less overhead than loading from the load library on disk.

Modules in the Pageable Link Pack Area are copied from the load library to the system's page data sets during system initialization. To be eligible for the PLPA a program must be reentrant. When a PLPA module is loaded, the system simply adjusts the user's page tables to point to the area of the page data set that contains the module. The pages are brought in to memory as required by demand paging. This greatly reduces loading overhead. Paging overhead is also reduced since the PLPA pages never have to be paged out.

Entries in the system catalog determine which modules are placed in the FLPA and the PLPA.

Load Library Member Formation Procedures

This section contains the job control statements used for creating the modules in the system Load Library. Most of the members are formed by running two jobs. The first job uses the Linkage Editor (/LOAD LKED) to create a load module file, from the object decks as input. The second job uses LDLIBE to copy the member from the load module file to the Load Library. Refer to the LDLIBE utility for an explanation of the options.

The LPA copies and main storage directory entries for these members will not be updated until the next time MUSIC is IPLed. Some of the jobs that follow use the REPL=T option. Using REPL=T means that an attempt will be made to use the same space occupied by the previous version. It may be advisable to rename old members and then use REPL=F if you are not sure the new module will work or if you do not plan to re-IPL MUSIC right away.

$CTL Module Formation

Refer to files $SYS:CTL.LKED and $SYS:CTL.APPLY for the two jobs required.

$FNPAK1 Re-entrant Subroutine Package

Refer to files $SUB:$FNPAK1.LKED and $SUB:$FNPAK1.APPLY for the jobs required.

$INP Module Formation (/INPUT Command)

Refer to files $SYS:INPUT.LKED and $SYS:INPUT.APPLY for the two jobs required.

$LST Module Formation (/LIST Command)

Refer to files $SYS:LIST.LKED and $SYS:LIST.APPLY for the two jobs required.

$OSTRAP Module Formation

Refer to files $CMP:OSTRAP.LKED and $CMP:OSTRAP.APPLY for the two jobs required.

$PRE Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
.ORG 1000
   ENTRY PRE
<---------object for PRE. (Source stored under code $SYS)
   NAME $PRE

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='$PRE',RENT=F,REPL=T
```

$PST Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
.ORG 1000
   ENTRY PST
<---------object for PST. (Source stored under code $SYS)
   NAME $PST

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='$PST',RENT=F,REPL=T
```

$PUR Module Formation (/PURGE Command)

Refer to files $SYS:PURGE.LKED and $SYS:PURGE.APPLY for the two jobs required.

$REN Module Formation (/RENAME Command)

Refer to files $SYS:RENAME.LKED and $SYS:RENAME.APPLY for the two jobs required.

$REX Module Formation

Refer to files $REX:REXX.LKED and $REX:REXX.APPLY for the two jobs required.

$ROUTING Module Formation (Route Name Table)

Refer to file $PGM:$ROUTING.S for instructions.

$SAV Module Formation (/SAVE Command)

Refer to files $SYS:SAVE.LKED and $SYS:SAVE.APPLY for the two jobs required.

$SIGNON Module Formation

Refer to files $SYS:SIGNON.LKED and $SYS:SIGNON.APPLY for the two jobs required.

$UPD Module Formation (/UPDATE Command)

Refer to files $SYS:UPDATE.LKED and $SYS:UPDATE.APPLY for the two jobs required.

$XMON Module Formation

Refer to files $LKD:XMON.LKED and $LKD:XMON.APPLY for the two jobs required.

APL Module Formation

See file $APM:MUSAPL.LKED for the link edit job. See file $APM:MUSAPL.PUTSYS for the LDLIBE job.

APLVSXEC Module Formation

ASM Module Formation

See files $CMP:ASM.LKED and $CMP:ASM.APPLY for the two jobs required.

ASMLG Module Formation

See files $CMP:ASMLG.LKED and $CMP:ASMLG.APPLY for the two jobs required.

BLKLET Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
.ORG 0000
<---------object for BLKLET. (Source stored under code $CMP)
   NAME BLKLET

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='BLKLET',RENT=F,REPL=T
```

CBMANIP Module Formation

Refer to files $VSM:CBMANIP.LKED and $VSM:CBMANIP.APPLY.

CMPMON Module Formation

Refer to files $SYS:CMPMON.LKED and $SYS:CMPMON.APPLY for the two jobs required.

COBOL Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
.ORG 4600
<---------object for COBOL. (Source stored under code $CMP)
   NAME COBOL

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='COBOL',RENT=F,REPL=T
```

DICT1 Module Formation (Spelling Dictionary)

Refer to *Chapter 12 - TODO Facilities* under the topic "Installing a System Word Dictionary in the PLPA".

DMSREX Module Formation

Refer to files $REX:DMSREX.LKED and $REX:DMSREX.LD for the two jobs required.

EDIT Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,PRINT,STATS,NOSEGTAB,MODE=OS
.ORG 1000
<---------object for EDIT. (Source stored under code $EDT)
 NAME EDIT

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='EDIT',RENT=F,REPL=T
```

EDITOR Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,PRINT,STATS,NOSEGTAB,MODE=OS
.ORG 1400
<---------object for EDITIO. (Source stored under code $EDT)
<---------object for EDITOR. (Source stored under code $EDT)
<---------object for EDTFIX. (Source stored under code $EDT)
<---------object for TSUSER. (Source stored under code $EDT)
<---------object for SUBMIT. (Source stored under code $PGM)
<---------object for PRINT.  (Source stored under code $PGM)
<---------object for QSCAN.  (Source stored under code $PGM)
<---------object for DUMMY.  (Source stored under code $EDT)
   NAME EDITOR

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='EDITOR',RENT=T,REPL=T
```

EDTDSP Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,PRINT,STATS,NOSEGTAB,MODE=OS
.ORG 0000
<---------object for EDTDSP. (Source stored under code $EDT)
 NAME EDTDSP

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='EDTDSP',RENT=T,REPL=T
```

EXEC Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
.ORG 1000
<---------object for EXEC. (Source stored under code $LKD)
   NAME EXEC

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='EXEC',RENT=F,REPL=T
```

FORTG1 Module Formation

See file $FG1:FG1.GEN for generation procedure.

IBCOM Module Formation

See file $LM1:LM1.GEN (Fortran Mod I library) or $LM3:LM3.GEN (Fortran Mod II library) for generation procedure.

IEFBR Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
.ORG 0000
<---------object for IEFBR. (Source stored under code $CMP)
   NAME IEFBR

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='IEFBR',RENT=T,REPL=T
```

IFOXnn Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
/INCLUDE $CMP:VSASM.LKED

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='IFOX00',RENT=F,REPL=T
NAME='IFOX01',RENT=F,REPL=T
NAME='IFOX02',RENT=F,REPL=T
NAME='IFOX03',RENT=F,REPL=T
NAME='IFOX04',RENT=F,REPL=T
NAME='IFOX05',RENT=F,REPL=T
NAME='IFOX06',RENT=F,REPL=T
NAME='IFOX07',RENT=F,REPL=T
NAME='IFOX11',RENT=F,REPL=T
NAME='IFOX21',RENT=F,REPL=T
NAME='IFOX31',RENT=F,REPL=T
NAME='IFOX41',RENT=F,REPL=T
NAME='IFOX42',RENT=F,REPL=T
NAME='IFOX51',RENT=F,REPL=T
NAME='IFOX61',RENT=F,REPL=T
NAME='IFOX62',RENT=F,REPL=T
```

IGIALL Module Formation:
IGIEXT Module Formation:
IGIGEN Module Formation:
IGIPAR Module Formation:
IGIUNF Module Formation:

See file $FG1:FG1.GEN for the generation procedure.

IISRENT Module Formation

See file $IIS:IIS.GEN for the generation procedure.

IKFCBLnn Module Formation

See file $GEN:COBOL4.GEN or $GEN.VSCOBOL.GEN for the generation procedure.

ILBOPRM0 Module Formation

See file $GEN:VSCOBOL.GEN for the generation procedure.

JOBONE Module Formation

Refer to files $SYS:JOBONE.LKED and $SYS:JOBONE.APPLY for the two jobs required.

LIBRARY Module Formation (/LIBRARY Command)

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
.ORG 1000
   ENTRY LIBCMD
<---------object for LIBCMD. (Source stored under code $SYS)
<---------object for MFINDX. (Source stored under code $SUB)
<---------object for MATCH. (Source stored under code $SUB)
<---------object for SSORT. (Source stored under code $SUB)
   NAME LIBRARY

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='LIBRARY',RENT=F,REPL=T
```

LKED Module Formation
LKEDPHS1 Module Formation:
LKEDPHS2 Module Formation:
LKEDPHS3 Module Formation:

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEARCH,MODE=OS    (note: option NOSEGTAB not used)
.ORG 0F80
   OVERLAY A
<---------object for SYSIO. (Source stored under code $LKD)
<---------object for LKDMON. (Source stored under code $LKD)
<---------object for SBLOPN. (Source stored under code $PGM)
   OVERLAY B
<---------object for PHASE1. (Source stored under code $LKD)
<---------object for PREP1. (Source stored under code $LKD)
   OVERLAY B
<---------object for PHASE2. (Source stored under code $LKD)
<---------object for PREP2. (Source stored under code $LKD)
   OVERLAY B
<---------object for PHASE3. (Source stored under code $LKD)
<---------object for PREP3. (Source stored under code $LKD)
   NAME LKED

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='LKED',SEG=2
NAME='LKED',RENAME='LKEDPHS1',SEG=3
NAME='LKED',RENAME='LKEDPHS2',SEG=4
NAME='LKED',RENAME='LKEDPHS3',SEG=5
```

LKEDEXEC Module Formation

See $LKD:LKEDEXEC.LKED and $LKD:LKEDEXEC.APPLY for the two jobs required.

LOADER Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
.ADD  004300    STRTUC
.ADD  001000    IBCOM#
.ADD  002394    FIOCS#
.ADD  002840    ADCON#
.ADD  003CE8    IHNERRM
.ORG  001000
<---------object for LLOADER. (Source stored under code $LDR)
   NAME LOADER

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='LOADER',RENT=F,REPL=T
```

OLOADER Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
.ADD  004800    STRTUC
.ORG  001000
<---------object for OLOADER. (Source stored under code $LDR)
   NAME OLOADER

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='OLOADER',RENT=F,REPL=T
```

PLI Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
.ORG 4600
<---------object for PLI. (Source stored under code $CMP)
   NAME PLI

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='PLI',RENT=F,REPL=T
```

QLOADER Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
.ADD  004300    STRTUC
.ADD  001000    IBCOM#
.ADD  002394    FIOCS#
.ADD  002840    ADCON#
.ADD  003CE8    IHNERRM
.ORG  001000
<---------object for QLOADER. (Source stored under code $LDR)
   NAME QLOADER

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='QLOADER',RENT=F,REPL=T
```

RLOADER Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
<---------object for RLOADER. (Source stored under code $LDR)
<---------object for SBLOPN. (Source stored under code $PGM)
   NAME RLOADER

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='RLOADER',RENT=T,REPL=T
```

SORT Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,MODE=OS   (note: NOSEARCH is not used)
.ORG 0000
<---------object for SRTMRG. (Source stored under code $CMP)
   NAME SORT

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='SORT',RENT=F,REPL=T
```

VSAM Module Formation

Refer to files $VSM:VSAM.LKED and $VSM:VSAM.APPLY.

VSAPL Module Formation

See file $APV:VSAPL.LOAD.LKED for the link edit job. See file $APV:VSAPL.LOAD.PUTSYS for the LDLIBE job.

VSBASCMP Module Formation

See file $BAV:VSBASIC.GEN for the generation procedure.

VSBASIC Module Formation

```
/FILE LMOD NAME(xxx) NEW(REPL)
/LOAD LKED
/JOB MAP,NOGO,NOSEGTAB,NOSEARCH,MODE=OS
.ADD  005000    CMPADR
.ORG 1000
   ENTRY SYSIO
<---------object for SYSIO. (Source stored under code $BAV)
<---------object for VSBMON. (Source stored under code $BAV)
   NAME VSBASIC

/FILE 1 N(xxx)
/INCLUDE LDLIBE
LIBE='SYST',IN=1
NAME='VSBASIC',RENT=F,REPL=T
```

VSBASLIB Module Formation

See file $BAV:VSBASIC.GEN for the generation procedure.

Link Pack Area Considerations

The Link Pack Areas are used to hold resident copies of transient system and compiler modules. These modules are read in from the system data set SYS1.MUSIC.LOADLIB.

The Fixed Link Pack Area (FLPA) and Pageable Link Pack Area (PLPA) are loaded at during system initialization based on the names given on RESPGM statements in the system catalog. These copies are subsequently used by the system avoiding the I/O overhead involved in loading them each time they are required. The LPA program can be used to display the current contents of the FLPA and PLPA.

FLPA modules are loaded into real memory and remain resident there. There is obviously not enough room in real memory to load all load library modules. High usage modules are the best candidates for the FLPA. By default, a number of system utility modules such as SIGNON and EDITOR are put in the FLPA. If an FLPA module is reentrant, then it can be used directly by the user program. This not only reduces the load

time but also cuts down on the swap and paging load. If it is not reentrant, a copy is made in the user region when requested.

PLPA modules are placed on the system's paging datasets during the initialization process. There is no real storage penalty for putting a module in the PLPA, but the module must be reentrant to be eligible. When a PLPA module is requested the system only need adjust the page table of that user. Any required page is then read in as required using the system page mechanism.

The utility LDCNTS can be used to determine how often a module is used. Most installations will find that the following are the most commonly used and therefore should be in the FLPA: $CTL, $PRE, $PST, $LST, EDIT, EDTDSP, EDITOR, and $SIGNON. Other good candidates are $OSTRAP, $XMON, EXEC, IBCOM, $REX, and DMSREX.

The following members must be in either the FLPA or PLPA: RLOADER, EDITOR, EDTDSP.

The member $ROUTING must always be in the FLPA. Even though it is reentrant it will not work in the PLPA as some nucleus routines refer to it.

Keeping VSBASCMP and VSBASLIB routines resident means that about 50K larger VS BASIC programs can be handled in the same size user region.

Keeping all C/370 modules resident has shown a marked reduction in compile times. The routine names are: EDCP, EDCO, EDCT, EDCXV, EDCX24, and IBMBLIIA.

The current sizes of modules on your system can be listed by running the LDLIST program. Non-reentrant modules in the FLPA will typically use 15% more space than indicated here due to space needed to hold the relocation information. The items marked RENT are reentrant modules and hence are suitable for the PLPA as well as the FLPA.

Chapter 22. System Programming

Overview

This chapter contains information of interest to system programmers who want to add new functions or make modifications to MUSIC/SP. Before undertaking a programming project there are some basics you should know.

The source for MUSIC is available on the source tape. This source is not only useful when you want to make modifications, but contains many valuable comments and programming examples. Some features of MUSIC internals that are not documented in manuals are described in detailed comments in the source modules themselves.

There are a number of choices available when adding a new program or command. The command could be written in REXX and interpreted directly from the REXX source. It could be written in a high level language or assembler and run from a load module or object deck. A load module could be run from the system Load Library or the LPA. A new command could be added to the system nucleus itself. The method chosen depends of the type of application and required performance characteristics.

You should keep track of any local modifications or additions so they don't get forgotten when applying maintenance or going to another release of the system. Also keep your own source separate from the base system source.

Naming Conventions For System Files

MUSIC system files follow a set of naming conventions based on the suffix used. The most common suffixes used are the following:

Source files end in .S

Object decks end in .OBJ

Files containing linkage editor control cards end in .LKED

Files containing load modules end in .LMOD

Files containing program data end in .DATA

Files containing macro definitions end in .M

Usage of $ Userids on MUSIC

All system files are stored under an ownership userid that starts with a dollar ($) sign. Files associated with different system functions are stored under different userids as follows. (For the latest table of $ userids see the file CATALOG.)

$ACT Contains the files associated with the session accounting program.

$ADM	Files associated with the ADMIN program for the system administrator.
$ADS	Files for the site's Ads Facility. These files are not distributed with MUSIC. They are created by the site.
$ADZ	Data files for the SAMPLE ADS (classified ads) Facility.
$APM	Contains the files associated with MUSIC/APL.
$APV	Contains the files associated with VS APL.
$AXF	Contains the files associated with VS Assembler (XF).
$BAV	Contains the files associated with the VS Basic compiler.
$BBS	The BBS utility program, which is used for HELP, ADS, CWIS, and other applications. Files with names starting $BBS:@CIO are a SAMPLE CWIS.
$BMK	Benchmark programs, for comparing and measuring machine and system speeds.
$CIC	Contains the files associated with IBM CICS.
$CMP	Contains the files associated with OS simulation and the VS Assembler.
$COD	Contains files associated with sign-on code utilities.
$CON	Contains the files associated with the CONF and CONFMAN utilities.
$CSL	Contains the Contributed Software Library.
$DEM	Contains the files containing demonstration programs and games.
$DW2	Contains the files associated with IBM DisplayWrite/370.
$EDT	Contains the files associated with the Editor.
$EMD	Contains files associated with the Mail facility operation.
$EML	Contains the source for the MAIL facility.
$EMM	Userid owning the mailbox containing the site's mail profile defaults that are copied to new mailboxes at creation time.
$EMP	Contains the files associated with the postmaster for the MAIL facility.
$EMS	Electronic Marks Submission facility.
$EMn	$EM1,$EM2,... Contain the outgoing mailboxes for the RDMAILER tasks that process incoming mail.
$FG1	Contains the files associated with the FORTRAN IV (G1) compiler.
$GDM	Contains the files associated with IBM GDDM.
$GEN	Contains the files used by system generation functions.

$GPS	Contains files associated with the GPSS product.
$HLP	Contains the files used by the HELP commands. Files with prefix "@ED." are used with the Context Editor HELP facility. Files with prefix "@GO." are used with the *GO mode HELP facility. Files with the prefix "@FC." are used with the MUSIC/FCS facility.
$IBB	Contains the files associated with IBM Basic.
$IBC	C/370 and C/370 Version 2, Compiler and Library.
$IIS	Contains the files associated with the Interactive Instructional Presentation System (IIPS) and with the Interactive Instructional Authoring System (IIAS).
$IMD	Contains the files associated with IBM GDDM IMD.
$INT	Contains the files associated with the Full Screen Interface (FSI).
$ITS	Contains the files associated with the indexed text searching utilities.
$IVU	Contains the files associated with IBM GDDM IVU.
$JCL	Model job control card files for the SUBMIT program.
$KRM	KERMIT file transfer (part of the CSL - Contributed Software Library).
$LDR	Contains the MUSIC loaders.
$LKD	Contains the MUSIC Linkage Editor.
$LM1	Files to generate the FORTRAN Mod I Library.
$LM3	Files associated with the FORTRAN Mod II Library.
$MAN	Contains the files associated with the online manuals.
$MBx	$MBA,$MBB,... Contains mailbox files for the MAIL facility.
$MCC	Contains CMS macros. (CMS is a component of VM.) These macros are needed to assemble IIS source.
$MCM	Contains the MUSIC system macros. They are used when assembling system source.
$MCS	Client-server programs for interfacing with PC workstations.
$MCU	Contains the MUSIC user macros (mainly OS macros).
$MDxx	($MDAA,...$MDZZ, $MD00,...$MD99) Mail data files.
$MEM	Contains files associated with the MEMO utility.
$MON	Contains some auto sign-on programs.
$PAN	Contains the files associated with the 3270 PANEL subsystem.
$PCW	Files associated with the Personal Computer Workstation (PCWS).

$PGF	Contains the files associated with IBM GDDM PGF.
$PGM	Contains the files associated with miscellaneous main programs.
$PIP	Contains the files associated with the pipe cmd.
$PLI	PL/I Optimizing Compiler and Libraries.
$PL2	PL/I Version 2 Compiler, Library and Test Facility.
$PLK	Contains the files associated with IBM GDDM (PCLK feature).
$PRT	Print queue and print files.
$PUB	Default anonymous FTP code.
$RDR	Contains the batch jobs submitted to the MUSIC internal reader.
$REX	Contains the files for the REXX command executor.
$RG2	RPG/370 Compiler and Library (5688-127).
$RPG	Contains files associated with the RPG II compiler product.
$SCR	Contains the files associated with MUSIC/SCRIPT.
$SRV	Contains the files for applying service and installing optional products.
$STP	Files associated with STATPAK. (Not distributed with MUSIC/SP.)
$SUB	Contains the files associated with the subroutine library.
$SYS	Contains the MUSIC system source.
$TCP	Contains the files associated with TCP/IP.
$TDO	Contains the files associated with the Time, Office, and Documentation Organizer (TODO).
$TPC	Files associated with the PC co-axial connected file transfer program.
$TUT	The TUTORIALS facility: tutorials on programming languages such as Rexx, Fortran, etc.
$VCT	Vector Facility Simulator (5798-DWF).
$VC2	Contains the files associated with VS COBOL II.
$VF2	Contains the files for VS Fortran, Version 2.
$VMR	Contains the reader queue used by the VMREADX utility.
$VP2	Contains the files associated with VS PASCAL.
$VSC	Contains the files associated with VS COBOL.
$VSF	Contains the files associated with the VS FORTRAN compiler product.

$VSM	Files associated with the VSAM (Virtual Storage Access Method).
$VSP	Contains the files associated with the VS PASCAL compiler product.
$WWS	Sample data files for Web server.
$XXX	Contains miscellaneous dummy files.
$001	Contains workspaces distributed with VS APL for lib 1. (Not distributed with MUSIC.)
$002	Contains workspaces distributed with VS APL for lib 2. (Not distributed with MUSIC.)
$003	Contains workspaces distributed with MUSIC for VS APL lib 3.

Source Key File

The file SOURCE.KEY contains a list of all the system files. All of these files can be found on the MUSIC/SP distribution tapes. Not all of these files are included in the Save Library when you install MUSIC/SP. For example source code is not loaded onto your system to save disk space.

An * beside a file name means that the file is on the tape but is not restored during system generation.

The LIBRARY command can be used to obtain a list of the files currently on your system. For example, type the command "LIBRARY $PGM:*" to obtain a list of all the files under the ownership userid of $PGM.

Use the MFREST utility to restore selected files from the source tapes as required. Consult the file CATALOG for sample usage of MFREST. This file also contains information on how to locate which tape specific files are on.

Modifying MUSIC's Nucleus

You must recreate the system nucleus to change the I/O configuration, modify a nucleus module or add a new module to the nucleus. The ADMIN facility can be used to make routine configuration changes. You should understand what goes on behind the ADMIN menus however, if you want to actually change the nucleus code.

Change the Source

If required, restore the source for the existing module from the distribution tape using MFREST. Source for the nucleus modules is stored under the userid $SYS. Make your changes using the editor. Use the FLAG command of the editor to identify your changes. This is done by issuing the following Editor command: "FLAG DEL=*/,COL=73". File the changed version using a different name from the original.

Assemble the Module

The file SYSASM is setup to assemble system modules. The following JCL is used to assemble the system module XXXX and save the object deck in the file XXXX.OBJ.

```
/FILE SYSPUNCH NAME(XXXX.OBJ) NEW(REPL)
/INCLUDE SYSASM
/OPT DECK
=XXXX
```

Modify the NUCGEN Job Stream

This can be done using the ADMIN facility or by simply editing the file $GEN:NUCGEN.JOB. The NUCGEN utility reads the base system object decks from the file $GEN:NUCLEUS. It then reads the device control cards and replacement object decks from the input data stream. As output, the NUCGEN program produces a file which contains an IPLable loader, the object decks for the system modules including the replacements, and the new device control cards. The output can be written to a file or to tape. The tape option is useful in that it provides a good backup copy of the nucleus. This output file can optionally be used as the base system input file for a subsequent run of the NUCGEN program.

To replace a system module an /INCLUDE statement pointing to the new object deck should be placed after the DEVEND statement. If more than one copy of that module is found, the first one is used.

If you are adding a new module to the nucleus, edit the file $GEN:MDLSYG and add)OBJ statement for the new module where you you want it to appear in the nucleus. It should be placed somewhere between the modules URMON and PROTND.

Install the Nucleus

Before installing the new nucleus make sure that you have a backup copy of the old nucleus on tape. If the nucleus was written to tape, attach a tape drive to the MUSIC virtual machine, shutdown MUSIC and IPL from the tape drive. If the nucleus was written to a file use the SYSGEN1 utility to install it directly. The ADMIN facility also uses this technique. Once the nucleus is installed reload the system and test out your changes. If you are making changes to system code, the tape method is preferred, since if something goes wrong you always have the tape from the previous NUCGEN as a backup. This is only true of course if you used the tape method the last time.

Backup copies of the MUSIC nucleus can also be kept in CMS files. Punch the file created by the NUCGEN job to a CMS machine, read it in and save it in a CMS file. To install the nucleus, punch a copy of the file to MUSIC and IPL the file from the virtual card reader. (Specify the no header option on the punch command).

Under VM, you can also setup a second copy of MUSIC to be used as a test system. This is very useful in debugging and testing system changes during normal working hours. If you are considering making major modifications to the nucleus this option should not be overlooked.

Modifying Applications, Utilities and Commands

This section gives a general overview of the steps required to modify or add applications, utilities or commands. The basic procedure involves three steps: get the source off the distribution tape if required; make the changes; install the new version.

Restore Source from Tape

You can use the ADMIN facility to restore source files as described in the *MUSIC/SP Administrator's Guide* under the topic "Restoring System Source Files".

Alternately, the MFREST utility is used to restore files from the distribution tapes. Consult the file CATALOG to find out what tape a specific file is on and sample jobs to restore them. When restoring groups of files with MFREST, never specify REPL=T unless you really mean it and always specify FIXUP=' ' unless you specifically want it to do name fixup. The files you restore from the distribution tape depend on what you want to do. If you have a lot of disk space you may want to restore all the files and have the convenience of having everything online.

Install the New Version

The procedure to follow here depends on the language used and on whether the program is run from a load module or from the load library.

If the program is written in REXX there is very little to do after the source changes are made since REXX runs directly from source. You can make REXX programs more efficient by removing comments and compressing the source using the REXDCOM and REXCOMP commands. This is only important for large programs or programs that are used frequently. (Remember to keep the original copy with the comments around). To install the new version simply replace the contents of the original file with the new REXX source.

Programs written in other languages must be compiler or assembled. They can then be made into load modules. Load modules can be executed directly or put in the Load Library. Files have been set up to assist you in performing these steps. The source files for many programs contain the control statements required to compile or assemble that program. Simply executing the file will automatically produce the object file. You create the load module by running the LKED file for the program. Sometimes an LD or APPLY file is also available to copy the load module to the system load library. (The load library is described elsewhere in this manual).

Suppose you are changing the AUTOPR utility.

$PGM:AUTOPR.S	Contains source. Executing this file compiles the source and produces the object program.
$PGM:AUTOPR.LKED	Executing this file creates the load module.

Making Changes with REPs

The Replace (REP) statement is a convenient method of making small changes to the MUSIC system without the need for reassembling the modules.

The REP statement is considered by the MUSIC loaders as a special form of a TXT statement found in object modules. The REP statement is normally inserted in the object module after the TXT statements and before the RLD statements. It will also work when placed just before the END statement of an object module if the information in the REP statement is not subject to relocation.

The first character on REP card can be punched by the combination of the 12-2-9 punches using the multi-punch key on a card punch. Alternately it can be entered while using the Context Editor's *hexadecimal input* feature. The 12-2-9 punch is hexadecimal 02. For example, type the following commands to place a hexa-decimal 02 in position 1 of a line:

```
xin;o 02;ain
```

The format of the REP cards is as follows:

```
card col.   0000000001111111111
            1234567890123456789

            .REP  aaaaaa sssccccc,cccc,cccc  comments
```

where:

col 1 is a 12-2-9 punch (hexadecimal 02)

col 2-4 are the characters 'REP'

col 7-12 is a 6 digit hexadecimal address relative to module and obtained from the far left column of an assembled listing. The address must be a multiple of 2.

col 14-16 is the control section id, normally 001. This number is hexadecimal. This can be verified by the SD number given in the External Symbol Dictionary on the first page of an assembled listing

cols 17-70 contain a maximum of 11 4-digit hexadecimal fields separated by commas.

Example:

```
12
2
9REP  000452  014700,0000 MAKE IT A NO-OP
```

REP cards can also be included in the NUCGEN jobstream after the DEVEND statement. A special)REP statement is used to indicate which module the REPs apply to.

```
)REP TCS
.REP  000100  010700 PUT NOPS IN TCS
.REP  000120  014700
)REP URMON
.REP  00024C  0147F0 FORCE BRANCH
```

")NOREP xxx" removes all reps previously made in module xxx.

SYSREP Utility

The SYSREP utility can be used to apply REPs to load library members. The format of the REP statement used by SYSREP is different from the one described above.

System Control Blocks and DSECTs

In understanding any system, knowledge of the layout of the system control blocks is very helpful. You can get an assembled listing of MUSIC's control blocks by assembling the module @MUSBK. This module is on the source tape under the userid $SYS. @MUSBK contains dsects of many of the system control blocks. It is provided to assist in interpreting core dumps and system code. Some bits and bytes may not be currently maintained. Avoid writing user code that refers to any bits or bytes directly as this may make it system level dependent. Most of the control blocks and DSECTs and mapped by macros. These system macros are stored on the user id $MCM.

Program Chaining and Multi-Tasking

MUSIC provides subroutines to control program chaining and multi-tasking. In program chaining a series of programs are run one after another. The parent program must finish before the child program starts. Multi-tasking allows the parent program to run in parallel with the child program it created. The parent may wait for the child to finish before continuing.

Program Chaining

The subroutines NXTCMD and NXTPGM can be used to schedule another program or a command when the current program finishes running. These are documented in the User's Guide. It is also possible to designate a particular program as a user's *ALWAYS* program either in the user profile, or dynamically using an option of the NXTPGM subroutine. When there are no other programs scheduled, rather than return the terminal to *Go mode, the system runs the *ALWAYS* program. This mechanism allows programs such as REXX and TODO to schedule the execution of other commands and programs but automatically regain control when the scheduled jobs are finished. *ALWAYS* programs are started from the beginning each time and must be responsible for saving information that must be preserved while scheduled commands and programs are running. Also, scheduling a new *ALWAYS* program replaces the previous one. When executing a chain of programs under the control of an *ALWAYS* program the /CANCEL command can be used to cancel the individual programs. The /CANCEL ALL command will remove the *ALWAYS*, program if it is flagged as cancellable. Obvious applications are in the area of providing restricted or subset views of the system.

Multi-Tasking

The NXTCMD subroutine can be called to create a new task. The underlying mechanism uses MUSIC's multi-session support to add a new session and start the new program running in that session. The parent task controls how the user views this new session. The calling sequence for NXTCMD is as follows.

```
    CALL NXTCMD(command,len,opts)

command - command or program to execute
len     - length of the command
opts    - options
```

The bits in the low order byte of the options parameter are defined as follows.

X'80' Set to 1 for multi-tasking request

X'40' Parent task will wait for child to terminate.

X'20' Delete the child task when the scheduled program finishes.

X'10' Display job startup messages.

X'08' Leave the parent task active on the terminal. The child task will run in the background until the user activates it with multi-session function keys or commands.

X'04' Don't let the user see the parent task while the child task is running. This is normally used in conjunction with X'40' to prevent the user returning to the parent task prematurely using multi-session function keys or commands.

X'02' Create the new task as a BTRM and disconnect it from the parent. The BTRM is deleted when the

new task finishes.

X'01' Make child task non-cancellable.

Defining BTRMs and Auto-Sign-on

In implementing various features of the MUSIC system there was a requirement to be able to run programs in the background without a terminal attached (BTRM), and to be able to run programs on terminals without keyboards (Auto-Sign-on). Since both these areas involve starting a program without user intervention, they have been addressed in the same way.

The problem of running background jobs was solved by introducing the dummy terminal class BTRM. A BTRM is specified as a device in the NUCGEN. This causes MUSIC to construct the appropriate control blocks during system initialization as if a terminal was defined at that address. The device address specified for the BTRM is NOT related to any physical device, but is only used as a mechanism to determine which code to sign on. By default the system should be set up to have BTRMs defined on 005 and 010-019. These are used to run system utilities that communicate with VM. The fact that there is also virtual punches defined on these addresses is not a problem.

The lowest (first) BTRM address (for example, 005 in the standard system) should be used for the System Log Message Server BTRM ($PGM:SYSLG). Any BTRMs (such as RDMAILER) that may send log records to the Log Server BTRM should come after it. This is so that at MUSIC startup time, the Log Server BTRM will start before any of the other BTRMs.

In the NUCGEN, Auto-Sign-on terminals are defined in the same manner as regular terminals. They also have the SIGNON option specified. As with BTRMs the device address plays a role in determining which code to sign on.

By default BTRM or Auto-Sign-on terminal is automatically signed on to the code $MONsss, where the subcode *sss* is the device address of the BTRM or terminal as specified in the MUSIC NUCGEN. If the code does not exist in the system code table the sign-on will fail and the BTRM or Auto-Sign-on terminal will not work. Alternately the userid can be defined in the file $PGM:AUTO.CONTROL. Each entry in the file contains a device address range and a userid. For example:

```
100-13F CWIS000
200-23F INFO000
250-250 PRT1000
```

Anything in the virtual address range 100-13F is signed on with the userid CWIS000, 200-23F uses INFO000, and 250 uses PRT1000.

During the sign-on process time limits, file space limits, privileges, terminal type, and operation default are all set from the user profile. Finally if an AUTOPROG is specified in the profile, and one should be, it is run. This allows the BTRM and Auto-Sign-on terminals to be signed on and start running the appropriate program.

The following steps should be taken when adding a new BTRM or Auto-Sign-on terminals to the system.

1. Decide on the device address and add the appropriate device specification statement to the NUCGEN job stream and create a new nucleus.

2. Allocate the code $MONsss, where *sss* the device address in question. When allocating the code set the appropriate password, time limits, terminal type, and privileges. The AUTOPROG parameter must also

be specified indicating which program is to be run. Use the CODUPD command "ADD $MON SC(sss)..." so that sss will be a subcode.

3. Set up the file to contain the auto-program defined in step 2.

4. Test the program by signing on to the code from a real terminal and verifying that there are no problems. This is very important since BTRMs have no terminals attached so you do not see any error messages when they are run in production. Auto sign-on terminals may be remotely located and again error messages may be difficult to access. Therefore testing the program before using it with the production BTRM or Auto sign-on terminal prevent problems later on.

5. Apply the new nucleus and verify the new BTRM of Auto sign-on terminal is functioning correctly.

BTRMs and Auto sign-on terminals respond to operator commands in the same way as do regular terminals. Normally the operator need not intervene, but under certain circumstances it may be necessary to cancel or restart one of the programs running on a BTRM or Auto sign-on terminal. Typically this situation occurs when the program or its parameters have to be changed, or the program is not functioning correctly. The following commands are recommended for these functions.

/CANCEL nnn	- the program is cancelled.
/RESET nnn	- the program is cancelled and restarted.

(nnn is the TCB number)

System Log Message Server BTRM (SYSLG)

Several MUSIC applications (for example MAIL and RDMAILER) need to be able to write many log and information records to a log file. Activity in such log files can be high - tens of thousands of records per day. In order to do this efficiently, these applications can send the log records as messages to a single log server task, via the Intertask Communications Queue (BPOOL buffers in memory).

The log server, running the program $PGM:SYSLG as a BTRM session, does all the work of managing the log files and writing the log records to the files. This is much more efficient, since each log file is only opened once, and many log records are batched together and written to disk with a single i/o operation. In addition, the log server can optionally redirect log messages to the operator console, to multiple files, to an id as e-mail, and to other programs. This is controlled by a definition file, $PGM:SYSLG.DEFINE. Log files can be defined so as to create a new file each day or month, with the date in the file name.

An application calls the SYSLOG system subroutine in order to send a log record to the log server. The application specifies an 8-character application name (e.g. MAIL and RDMAILER use the application name "MAIL"), a priority or class number (normally 0), and the text of the message (1 to 1024 bytes). Refer to the source file $SUB:SYSLOG.S for details. The calling program or user must have at least the SYSCOM privilege. The log server receives the message (at some later time up to 3 minutes after the SYSLOG subroutine was called) and uses the application name and priority/class number to route the message, according to the definitions in $PGM:SYSLG.DEFINE.

The format of the definition file $PGM:SYSLG.DEFINE is described in file $PGM:SYSLG.DEFINE.DOC. The define file is read once when the log server BTRM starts. If you change the define file and you want the change to take effect now, rather than at the next MUSIC IPL, you must stop the BTRM (by operator command "REPLY n STOP" where n is the TCB number - see below) and restart it (by "RESET n"). If the define file is missing or contains an error, the log server automatically appends all log records to file $000:@SYSLOG.MISC.

When setting up the log server, the lowest (first) BTRM address (for example, 005 in the standard system) should be used for it. Any BTRMs (such as RDMAILER) that may send log records to the Log Server BTRM should come after it. This is so that at MUSIC startup time, the Log Server BTRM will start before any of the other BTRMs, and no log records will be lost. Also, when shutting down MUSIC, in order not to lose any log records, stop BTRMs (such as RDMAILER) that generate log records, then stop the log server (see below), and finally shut down MUSIC.

The userid for the log server BTRM should be created by a command similar to the following, entered in the CODUPD utility. "xxx" is the 3-digit hex BTRM address, as specified in the MUSIC nucleus configuration. For example, in the standard MUSIC it is 005.

```
ADD $MON SC(xxx) PW(*NOLOGON) PROG($PGM:SYSLG) -
  PRIME(NL) NONPRIME(NL) BAT(0) DEF(NL) XSES(10) -
  FILES MAINT LSCAN SYSCOM -
  NAME(BTRM FOR SYST LOG SERVER)
```

The following MUSIC operator commands are used to communicate with the log server. "n" is the TCB number of the log server, which can be obtained by the command "enqtab syslog" entered in a MUSIC terminal session.

REPLY n HELLO	To verify that the log server is running, and to cause it to process any accumulated log messages.
REPLY n STOP	To cause the log server to process any accumulated log messages, then shut down the log server program. WARNING: After the log server is shut down, any log records generated by application programs will be discarded.
RESET n	To restart the log server, after stopping it. This command should only be used after "REPLY n STOP" has been used to shut down the log server.

MFIO Subroutine Interface

Two subroutines, MFIO and MFACT, are provided to give high level languages access to MUSIC's I/O interface. Before attempting to use these routines you should review the information on the assembler and macro interface in chapter 19.

MFACT - Open/Close a file

```
CALL MFACT(rc,'type options.','shortfilename ')
CALL MFACT(rc,'type options.',-1,'longfilename ')
```

rc	File system return code.
type	Operation type (OPEN, CLOSE, EXTRACT).
options	Options dependent on the type of operation (e.g. OKNEW OKOLD WROK RDOK APPOK REPL RENAME etc..) The option string is terminated by a period.
shortfilename	The file name (22 characters) terminated by a blank unless the file name is the maximum length, 22 characters long.
longfilename	The file name (64 characters) terminated by a blank unless the file name is the maximum length, 64 characters long.

Examples:

```
CHARACTER*22 FNAME1
CHARACTER*64 FNAME2
FNAME1(1:)=' '
FNAME2(1:)=' '
FNAME1(1:9)='ABCD:TEMP'
FNAME2(1:15)='ABCD:\WORK\TEMP'
CALL MFACT(IRC,'OPEN RDOK OKOLD.',FNAME1)
CALL MFACT(IRC,'OPEN RDOK OKOLD.',-1,FNAME2)
```

MFIO - Read/Write a file

```
CALL MFIO(rc,'type options.',pos,len,buf)
```

rc	File system return code.
type	Operation type (IO, UIO, FSIO).
options	Options dependent on the type of operation (e.g. RD WR WEOF REW FILL TRUNC RBA etc..) The option string is terminated by a period.
pos	RBA value used if RBA option is specified. Block number for UIO requests.
len	Length of buffer.
buf	I/O buffer.

The string '*.' can be used in the second parameter as shorthand for 'IO RD FILL.'. The string '$.' can be used for 'IO WR TRUNC.'.

MFGETU/MFSETU Subroutines

```
CALL MFGETU(unit)
CALL MFSETU(unit)
```

unit is the integer internal unit number.

When MFACT is called to open a file the internal unit number assigned to that file is stored in the basic request block used by MFIO and MFACT. Once the file is opened, all that is required during I/O operations is this unit number, the buffer address and the length. However, if MFACT is called again to open a second file, the control blocks are reused and the old unit number is lost. The subroutines MFGETU and MFSETU allow the programmer to have a number of files open at the same time by saving the unit number after opening a file (MFGETU) and resetting it whenever an I/O operation is to be done on that file (MFSETU).

MFIO Common Blocks

The following common blocks names are assigned to the MFIO control blocks used by these routines. Detailed information on the contents of these control blocks can be found in Chapter 20.

MFARGX	Copy of MFIO basic request block
MFTAG	Tag information
MFINFO	Infin/infout block
MFHINF	User information block for 16 character userids
MFUINF	User information block
MFEOFP	End of file pointers
MFBUPN	File backup number
MFLIBR	Argument for library request
MFUCTL	User control record
MFEXTN	File extent information

MFFSAR	FSIO argument
MFXINF	Usage information block
MFPHYS	Actual length read, RBA information
MFRNAM	Returned file name from MFACT

MFIO tracing

You can get full tracing of MFIO and MFACT subroutine activity. To activate this feature a /FILE statement with a ddname of MFTRACE is required. For example:

```
     /FILE MFTRACE N(GORK) NEW(REPL) RECFM(VC)
or   /FILE MFTRACE PRT
```

The tracing information looks as follows:

```
MFACT AT AAAAAA RC=RR U=UU REQ= PARM TEXT/XXXXXXXXXX
```

Where *aaaaaa* is the address from where MFACT or MFIO is being called from. *rr* is the return code. *uu* is the internal unit number. "parm text" is the keyword literal text passed in the call. *xxxxxxxxx* is the resulting basic user request block.

Each type request provides additional useful information.

OPEN and EXTRACT	- file name and block INFIN
CLOSE	- blocks EOFPT and TAG
IO and UIO	- block PHYS and 40 bytes of the I/O buffer

Enqueue/Dequeue Facility

The MUSIC subroutines ENQ and DEQ allow the user to gain shared or exclusive control of a resource (by calling ENQ) and subsequently to release control of the resource (by calling DEQ). The QUERY subroutine allows you to check if the resource has been enqueued. This facility is used to coordinate the concurrent use of resource by several users.

The resource is identified by an arbitrary 8, 10 or 24 character name, of which the first character must be a letter A to Z. Any number of users can have simultaneous shared control of the resource, but only one user at a time can have exclusive control of the resource. A user's control of any resource is automatically released when the user's job terminates. Each user may have control of no more than 2 resources at any one time. Improper use of the enqueue/dequeue facility may cause the user's job to be aborted.

A privileged system programmer may display the contents of the system's enqueue table by executing the ENQTAB utility, described in Chapter 17 in this manual.

The calling sequences are as follows:

```
CALL ENQ(TYPE,NAME,RETCD1)

CALL DEQ(TYPE,NAME,RETCD2)

CALL QUERY(TYPE2,NAME,RETCD3)
```

TYPE Integer specifying type of control and length of resource name:

1 request for shared control, 8-byte name
2 request for exclusive control, 8-byte name
3 request for shared control, 10-byte name
4 request for exclusive control, 10-byte name
5 request for shared control, 24-byte name
6 request for exclusive control, 24-byte name

In the call to DEQ, TYPE should be the same as in the corresponding call to ENQ.

TYPE2 Integer specifying the length of resource name:

0 request for 8-byte name
1 request for 10-byte name
2 request for 24-byte name

NAME Maximum length of resource name (8, 10, or 24 bytes long). The first character must be A to Z for a non-privileged user.

RETCD1 Integer return code from ENQ:

0 successful enqueue (the user has been given control as requested)
1 resource is not available (shared control was requested and a user already had exclusive control, or exclusive control was requested and a user already had shared or exclusive control)
2 error

RETCD2 Integer return code from DEQ:

0 successful dequeue (control has been released)
1 the user did not have control
2 error

RETCD3 Integer return code from QUERY:

0 resource is not enqueued
1 resource is enqueued (shared)
2 resource is enqueued (exclusive)
3 error

Unbuffered Tape I/O

Normally MUSIC will automatically buffer I/O to tape. This can be overridden by using RECFM(U) on the /FILE statement. When this is done, each read or write will be unbuffered. An automatic end-of-file will not be done for I/O done this way. The TAPUTIL utility uses this facility to handle variable length unspecified blocks. The subroutine TPIO can be used to issue the file system calls to do I/O in this manner. (The object deck for TPIO is in the file $PGM:TPIO.OBJ, not in the subroutine library). The calling sequence is as follows.

```
CALL TPIO(U,O,R,BUF,LEN)
```

U	Unit number of the tape.
O	Operation code. 0 - Read (max Length = 32760) 1 - Write 2 - Write EOF 3 - Read (max length = "LEN")
R	Return code. -1 - End of file 0 - Normal >1 - MFIO return code
BUF	Buffer address.
LEN	Length. On a read this value is set.

Logical Device Interface

This interface can create a VM terminal session using logical device support and thus allow a user to connect to other virtual machines directly from a MUSIC session. This gives the MUSIC user access to applications that are not running on his MUSIC machine. These are referred to as remote applications. The interface can control what goes on in the session it creates or turn control over to the MUSIC user's keyboard. In the typical situation a MUSIC application would create a session, issue the appropriate command sequences to connect the user to a remote application and then turn control over to the user. When the remote application terminates, the MUSIC application regains control and issues the commands to log the user off the remote system.

The interface consists of a suite of subroutines that are used by the application to create, manage, and terminate the remote session. In addition to providing a terminal session connection the interface can also do file transfer using the 3270 file transfer protocol.

CALL LDLOG(n)

Log operations within logical device support to the user file LDEV.LOG. Logging is off by default.

0 turn logging off
1 turn logging on

CALL LDINIT

Create a logical device.

CALL LDECHO(n)

Echo session on users terminal. The user can see what's happening but can't use the keyboard.

0 turn echo off
1 turn echo on

CALL LDTERM(kseq,lk,row,col,hseq,lh)

Pass control to the user's terminal.

kseq Keyboard escape sequence. Any commands prefixed by this are passed to the local MUSIC system for processing. The following exceptions are handled directly by the LDEV routine.

RECeive	- receive a file from the remote host
SEND	- send a file to the remote host
LOG	- log interrupts to LDEV.LOG
NOLOG	- stop logging interrupts
END	- return to controlling program (also QUIT and QQ)

lk Length of kseq
row row to scan for hseq
col column to scan for hseq
hseq Host escape sequence. When this character sequence is located in the specified location on the screen control is returned to the calling program. This is useful in regaining control to perform automatic logoffs. The scan follows the same rules as LDWAIT.
hl Length of hseq

CALL LDPUT(row,col,data,len)

Put data into a modifiable field. No checking is done on the validity of the data or the field.

row row number
col column number
data data
len length of the data

CALL LDCUR(row,col)

Set the cursor address prior to pressing a function key.

row row number
col column number

CALL LDKEY(key)

Press a function key.

key integer value representing the key

0	: enter
1-24	: PF key
-1	: PA1
-2	: PA2
-3	: PA3
-4	: CLEAR

CALL LDENT(data,len)

Put data in the input area and press enter. This is useful for applications that are not field dependant,

data data
len length of the data

CALL LDGET(row,col,buf,len)

Get data from the in core screen buffer. This need not be used on a field basis. The routine will pick out whatever part of the buffer you want. Note that currently keyboard input is not maintained in the buffer, only host transmitted data can be retrieved.

row row number
col column number
buf local buffer to receive the data
len length of the data

CALL LDWAIT(row,col,data,len,secs,rc)

Wait for a certain string of text to appear on the screen. This is useful in synchronizing events and preventing sending data before the host session is ready for it.

row row number
col column number
data data
len length of the data
secs The number of seconds to wait before returning to program empty handed.
rc This is non zero if the text was not found within the specified limit.

Note: If col=0 the screen is scanned starting from the specified row. If row and col are zero the entire screen is scanned.

CALL LDWINT(secs)

This causes the support routine to wait for an external interrupt or so many seconds, whatever comes first. It can also be used to force the interface to process any interrupts that are currently in the stack if the time limit is set to zero.

secs number of seconds to wait

CALL LDCLR

Clear the screen buffer.

CALL LDSEND('lname rname opt',len,flag,rc)

Send a file to the remote system.

lname local file name
rname remote file name
opt file transfer options
len length of first parameter
flag 1: display file transfer monitor.
rc return code

CALL LDRECV('lname rname opt',len,flag,rc)

Receive a file from the remote system.

lname	local file name
rname	remote file name
opt	file transfer options
len	length of first parameter
flag	1: display file transfer monitor.
rc	return code

Options If the remote system is a CMS system the the options must be separated from the file names by an open bracket "(". The options are the same as the 3270 file transfer program.

Note: The user can invoke the file transfer facility from the keyboard using the %RECEIVE and %SEND commands. ("%" is the escape character) The parameters on these commands are in the same format as the first parameter of the subroutines. For example:

```
%rec music.file cms file (crlf
%send music.file tso.file crlf
```

This sample FORTRAN program listed below is the source for the VM program that is documented in the *MUSIC/SP User's Reference Guide*.

```
CALL LDECHO(1)
CALL LDINIT
CALL LDWAIT(1,2,'VIRTUAL',7,3)
CALL LDTERM('%',1,0,0,0,0)
CALL EXIT
END
```

Logical device routines can be called from REXX, however, the format of the CALL statement in REXX is different from FORTRAN. The following is the REXX version of the sample program above.

```
/INC REXX
      CALL LDECHO  1
      CALL LDINIT
      CALL LDWAIT 1,2,'VIRTUAL',7,3,'RC'
      CALL LDTERM '%',1,0,0,0,0
```

Note that the 'RC' parameter must be present on the call to LDWAIT. Unlike FORTRAN, in REXX all parameters must be specified.

For a more elaborate REXX example see the file $PGM:LDEV.REX.

SIGNON Subroutine (REXX)

The following is a sample REXX application that can be used to access other systems in an automated way. The target could be on the same system, or accessed via VM-Passthru. A specific ID/Password can be used, or a 'shared' id, selected from the shared table can be utilized.

Calling Sequence: CALL SIGNON 'id' 'type' 'system' 'ipl' 'MSG(x)' 'TDISK(xxxx,yyy,z)' 'application'

Parameters:

The parameters are positional.

id SHARED, select ID/Password from table. UNIQUE(id/password) use the supplied ID/password.

type — LOCAL, local VM system is target system. PVM(node), target system accessed via VM-Passthru.

system — VM, target is a VM system. MUSIC(VMID), target is a MUSIC system running in the virtual machine VMID. the ID/Password used MUST be specified via the unique parameter.

IPL
NOIPL — If 'NOIPL' is used, then logon to VM id will be done with the 'NOIPL' option. When the first CP read is issued, the an IPL CMS will be done, and for the next read an 'acc 191 a (noprof' will be used, so CMS goes to command mode. If 'IPL' is specified (or is not specified), then an IPL of CMS is done and the profile exec (if it exists is executed).

msg — specifies the message levels that will be produced from the SIGNON call.
0 default if not specified. Issue the message about starting the automated sign on process.
1 no messages (except errors are displayed)
2 display a message at each major step of the sign on process
3 turn on echoing, let caller see all screens.

TDISK — allocate a TDISK, using:
xxxxx - the number of blocks
yyy - the virtual disk address
z - the file mode
(TDISK is only allowed if the SYSTEM is VM.)

application — specifies the name of application that is running. This parameter is not type-checked but is used to match the application name in the Shared ID file. If application is not specified, a Shared ID without an application name will be used. If application has a value, application must match a SHARED ID's application name for the SHARED ID to be used.

After processing, return will be made to the caller with echoing OFF and control has NOT been passed to the users terminal. If MSG(3) was specified, then echoing has been turned on. It is up to the caller to issue a LDTERM call. When going through Passthru, the LDTERM would normally be of the form:

```
CALL LDTERM '%',1,6,12,'APPLICATION TERMINATED',22
```

On return, if 'shared id' processing has been requested, the ID used will be returned in the RESULT. When the logical session returns to REXX, you MUST issue the following call, to free up the 'shared id'.

```
CALL DEQ 6, userid ,'RC'   /* remove name from enqtable */
```

where userid is the ID provided in the RESULT variable.

3270 Full-Screen I/O Interface.

Using PANEL is the best way to implement 3270 applications. This section describes the FSIO interface that PANEL uses to communicate with 3270 terminals and PCs.

To use the FSIO interface, the application program places output data and orders in a buffer, then issues an FSIO request. This sends the output data to the screen. The program is then delayed until the user presses an action key, at which time the input data (from a READ-MODIFIED operation) is transferred to the program's input buffer. The program then resumes execution and processes the data from the modified fields. This write-read sequence can be repeated many times. The format of the input and output data streams is documented in the publication *IBM 3270 Information, Display System Data Stream Programmer's Reference*.

The application program can address the screen by using a 1-byte row number followed by a 1-byte column number, instead of the 2-byte screen address used by the hardware. The FSIO interface automatically does the required conversion.

FSIO Assembler Language Interface

This interface allows full screen I/O to 3270 terminals. It does this through the MFIO SVC (MFREQ). Each request can result in a WRITE, READ, WRITE/READ or control operation. After setting up the arguments and buffers the MFREQ SVC is issued to perform the I/O. The system then takes the data from the WRITE buffer and places it into the system full screen buffer pool and stops the user's time slice. The user may be swapped out at this time. The data is then written to the terminal using the system buffers. When any action key is struck at the terminal, a read modified is done, the data placed in a system buffer and the user made eligible for user region service. When the user program next gains control, the system will transfer the data to the user's READ buffer.

Arguments

Three MFIO arguments are used.

```
Basic req block    -  a1   MFARG FSIO,U=9,FSARG=a2,PHYS=a3
                           MFGEN

FSARG block        -  a2   MFVAR FSARG,PICT=Y

PHYS block         -  a3   MFVAR PHYS,PICT=Y
```

The basic request block contains control information for MFIO, pointers to the other args and return codes.

The FSARG block contains control information, data lengths and buffer addresses. It expands as follows.

```
          4         4         4          4          4
     | control | wrt len | wrt adr | read len |read adr |

control byte 0 - x'80'  -  write erase
                 x'40'  -  erase all unprotected (after read)
                 x'20'  -  write structured field
                 x'10'  -  asynchronous read
                 x'08'  -  skip write operation
                 x'04'  -  skip read operation
                 x'02'  -  user supplies own WCC.
                 x'01'  -  convert data (x,y -> 3270 form)
                 x'00'  -  normal WRITE/READ operation.
                 x'0C'  -  immediate read modified
                 x'0D'  -  immediate read buffer

control byte 1 - x'80'  -  no data compression on this request
                 x'40'  -  no data compression on this request
                           and all following requests in this job
                 x'20'  -  no APL conversion on this request and
                           all following requests in this job
                 x'10'  -  remember x'40' and x'20' option bits
                           from this control byte, but do not do
                           any I/O operation
```

```
                  x'08'  -  no translation for 3270 model 5 terminal
                            (27 by 132 size screen)
                  x'04'  -  ignore screen not initialized error
                  x'02'  -  reserved
                  x'01'  -  reserved

control byte 2 - x'nn'  -  Overriding Channel Command

control byte 3 - reserved.
```

The actual length of the data moved to the users read buffer is saved in the first word of the two word PHYS block.

Return codes

The return code is stored in byte 8 of the basic req block. A return code of 0 is normal. If any error has been detected this code is set to 2 or 11 and in this case bytes 9 and 10 should be checked to find out what caused the error. Values are as follows.

```
Byte 9.

  x'80'  -  not a 3270.
  x'40'  -  bad screen.  Screen is not setup for full screen I/O.
            Probably caused because no full screen initialization
            was done or some non-full screen I/O was since then.
  x'20'  -  bad length on write (0 or too big)
  x'10'  -  bad length on read (0)
  x'08'  -  bad screen addr (conv x,y -> 3270 addr)
  x'04'  -  buffers not available for write (retry later)
  x'02'  -  reserved
  x'01'  -  no data from read. Test request was hit.

Byte 10.

  x'80'  -  input truncated (user buffer too small)
  x'40'  -  bad data stream from terminal (I/O error on read)
  x'20'  -  I/O error on write
```

Data Conversion

The interface can optionally perform some data conversion for you. This is requested by setting the X'01' bit in control byte 1. If this is not set raw 3270 data is expected from, and passed back to the program. Data conversion is done as follows.

Write buffer — The buffer is scanned for any occurrences of SBA, RA or EUA. In each case the next two bytes are converted from X,Y to 3270 addresses. An error condition is set if these are not on the screen.

Read buffer — Data returned to the user buffer will be in the format...

```
   aid,x,y     x,y,len,data     x,y,len,data...etc

   cursor         field 1          field 2
```

The length in front of each field is a halfword.

Note: Data conversion is not performed on data associated with a WRITE STRUCTURED FIELD command.

Notes on Usage

In addition to the standard write/read request a number of options can be selected using the control bytes. Either the write or the read part of the operation can be skipped. This allows programs to handle output and input as logically distinct. The system will automatically clear the input fields after a read if the erase all unprotected option is selected. This may be useful in data entry applications.

During a normal read operation the application program is stopped, and the system waits for an attention interrupt before doing the read and rescheduling the application. The asynchronous read option allows the application to continue running while the system is waiting for the attention interrupt. When the interrupt occurs, the system does the read and sets the post code to ATTN. The application can test the post code using the PSTCOD subroutine. If the application now issues a read request is given the data from the asynchronous read. If it issues a write, the read data is purged. The application could also choose to issue an immediate read buffer or read modified request at this time.

On completion of a full screen read any action key and associated modified fields are returned to the program with the exception of the TEST REQ key which is reserved as an escape mechanism and puts the terminal in ATTN mode so that BREAK time commands can be entered. In this case no data is returned and the error flag (byte 9 in MFARG) is set to x'01'.

Full screen I/O and regular I/O may be inter-mixed if certain rules are followed. Full screen I/O must begin with either a WRITE ERASE or a WRITE STRUCTURED FIELD. (If this is not the case a BAD SCREEN condition will be returned to the program.) From then on any full screen operations can be used. Any regular I/O from either the program or the system will automatically reset the screen to MUSIC mode and full screen I/O must be re-initiated.

The return flag should always be checked for error conditions.

Buffer Pool

The interface utilizes the terminal buffer pool which is allocated during system initialization. The BPOOL parameter in the NUCGEN program is used to inform the system of how many buffers should be allocated. Each buffer is 516 bytes long, 512 bytes for data and 4 for a chain pointer. These buffers are allocated to a particular terminal only when necessary. For example suppose a program wanted to write 1.2K of data to a terminal and read the response. FSIO would request a chain of three buffers from the pool, move the data there and schedule the write. When TM3270 completes the write it frees the buffer chain and waits for an attention interrupt. When that occurs, since the terminal could possibly transmit up to 2K of data during the read, TM3270 gets a chain of 4 buffers from the pool to do the read, unused buffers are freed immediately. The buffers that actually contain data cannot be freed until the application program is again swapped into the user region.

If a program issues a write request that cannot be accomplished due to lack of buffers, it is informed by a return code, and should re-try the operation after waiting some time. If this condition occurs often, systems personnel should allocate more buffer space by increasing the BPOOL parameter and doing a NUCGEN.

If a read operation cannot be done due to lack of buffers the system will automatically re-try it later.

FSIO Subroutines

The following subroutines can be used instead of the assemble macro interface.

WBUF1 This routine places 3270 orders and data strings into the output buffer. The buffer is contained in the common block /BUFCOM/.

FSIO After the output buffer is complete, this routine is called to issue the write-read request.

GETAID After the read is complete, this routine examines the input buffer to find which action key was pressed and the row and column position of the cursor. The input buffer is in common /BUFCOM/ (occupying the same storage as the output buffer). GETAID also does initialization required for calls to GETFLD.

GETFLD After the read, each call to this routine returns the next screen field in the input buffer, i.e. the next modified field. An alternate return is taken when there are no more fields. GETAID must be called before the series of calls to GETFLD.

TRANSL This routine translates (using the TR machine instruction) a string of characters according to a specified translate table. It should be used to remove screen control characters from output data strings, and to translate input to upper case when necessary. This routine is described in the *MUSIC/SP User's Reference Guide*. It is recommended that characters X'00' through X'3F' and X'FF' be translated to blanks in output data strings.

Calling Sequences

```
CALL WBUF1(N1,N2,...,LEN,DATA,&n)
```

N1,N2,... From 0 to 10 integer values, used for specifying order bytes (SBA, SF, etc.), attribute bytes, and row and column numbers. The 4th byte of each argument is placed into the buffer.

LEN The length of an additional character string (in DATA) to be placed into the buffer. LEN=0 indicates a null string. If LEN is -1, the string in DATA is ended by $$.

DATA Additional characters to be placed into the buffer, after any bytes specified by N1,N2,...

&n Alternate return taken if there is no more room in the output buffer.

Note: BUFPTR in common /BUFCOM/ must be set to 0 before a series of calls to WBUF1.

```
CALL FSIO(TYPE,WRBUF,WRLEN,RDBUF,MAXRD,RDLEN,RETCOD)
```

TYPE This 4-byte integer value contains the FSIO control bytes that specify the type of operation that is to be performed. The bit settings correspond to those defined in the control byte fields for the assemble macro interface. The bytes are reversed however. So the low order byte of TYPE corresponds control byte 0 and the next one to control byte 1.

WRBUF The write buffer.

WRLEN The number of bytes of data to be written.

RDBUF The buffer to be used for the input operation. If WBUF1 and GETAID/GETFLD routines are used, the read buffer is in common /BUFCOM/ (see below) and coincides with the write buffer.

MAXRD The size of the read buffer, i.e. maximum read length.

RDLEN This is set by FSIO to the length of data actually read.

RETCOD This 4-byte integer value contains the return code from the FSIO request. 0 indicates a successful operation. If an error condition occurs, RETCOD is in the format X'0Bxxyy00' where *xx* and *yy* correspond to bytes 9 and 10 of the basic I/O request block. The meanings of the individual bits are described in the assembler macro interface section.

```
CALL GETAID(KEYHIT,CROW,CCOL)
```

KEYHIT Set to an integer value indicating which action key was pressed:

0	ENTER key
1 to 24	Program function key
-1	PA1 key
-2	PA2 key
-3	PA3 key
-10	CLEAR key
-99	Some other key, or an error condition.

CROW Set to the row number of the cursor at the time of the read.

CCOL Set to the column number of the cursor at the time of the read.

Note: The variable RDLEN in common /BUFCOM/ must be set before calling GETAID. Also, GETAID must be called before a series of calls to GETFLD.

```
CALL GETFLD(FROW,FCOL,POS,FLEN,&n)
```

FROW Set to the row number of the first character of the modified field, i.e. the character following the attribute byte.

FCOL Set to the column number of the first character of the modified field.

POS Set to the position of the start of the data for the modified field in the read buffer. The first character of the buffer is considered to be position 1. POS may be set to 0 if FLEN is 0.

FLEN Length, 0 or more, of the data for this field.

&n Alternate return taken when there are no more modified fields.

Common Block Used

COMMON /BUFCOM/ BUFSIZ,BUFPTR,RDLEN,BUF

Routines for Scanning the Code Table

Code $COD contains some subroutines which can be called by privileged users to *scan* the code table, i.e. process all the records, one by one, in index order. For example, they could be used to change a particular code record field or option bit for all codes, or for all codes satisfying some condition. They could be used to print a code table listing in some desired format.

Refer to the comments in files $COD:NXTCOD.S and $COD:CDGETU.S for detailed descriptions.

NXTCOD	Returns the next code record from the table. This routine should be used when none of the records will be changed.
NXTCDA	Similar to NXTCOD, but only the next userid (7-byte code/subcode) and a pointer to the code record are returned. This routine should be used, with CDGETU and CDUPDT or CDFREE, when some of the records will be changed.
CDGETU	Gets a code record for updating. Enqueues for exclusive control of the code record. Either routine NXTCDA or the code search SVC (or CDSVC routine) must have been used previously to get the (pseudo) disk address of the code record.
CDUPDT	Writes an updated code record to disk and dequeues. The record must have been gotten by CDGETU.
CDFREE	Releases control of a code record (dequeue). This routine must be called instead of CDUPDT if a record obtained by CDGETU is not to be changed.

Defining a First-Time Program

The parameter FIRST(n) on a CODUPD ADD or CHANGE command can be used to define the ID number *n* (1 to 255) of a special program that is to be run when the user signs on for the first time. For example, the first-time program could prompt for information about the new user and store it in a log file.

The file names for the ID numbers must be defined by modifying a table in system module SIGNON (member $SIGNON in the Load Library, source in code $SYS). The file name must also be added to the table in module LOOKUP (source in $SYS), since the program must be privileged, with at least the CODES privilege.

Once the first-time program has done its processing, it must zero the ID number in the user's code record, to prevent the program from being invoked the next time the user signs on. Also, it must schedule the user's normal auto-program, if any. This can be done by calling subroutine NOPGM1. See the comments in file $COD:NOPGM1.S for details.

The system is distributed with one first-time program. It has an ID number of 1. Its source is stored under the name $PGM:FIRSTTIME.PGM.S.

Defining an Alternate System Catalog

Normally MUSIC will read the system catalog from the data set called SYS1.MUSIC.CATALOG. This can be changed by the use of the =CATxxxx special option at IPL time. The alternate name will be SYS1.MUSIC.CATxxxx in this case.

The alternate catalog data set must first be allocated on MUSIC's IPL volume, and formatted. Then the data is written to it by using the EDTCAT utility program.

Submitting Jobs Automatically

The AUTOSUB utility can be used to automatically submit batch jobs through VM at certain times of the day and even on certain days of the week. Examples of use include submitting the system usage information statistics programs IOTIME and COUNTS. Refer to the writeup of AUTOSUB in *Part IV - Utilities*.

Miscellaneous System Subroutines

CORZAP

This routine modifies (zaps) a string of bytes anywhere in main storage. The VIP privilege is required.

Calling Sequence: CALL CORZAP(frmbuf,len,loc)

Arguments:

frmbuf	Replacement bytes.
len	Number of bytes to be replaced (any value .ge. 0).
loc	Starting addr of main storage to be replaced.

FETCH

This routine fetches a string of bytes from main storage. Non-privileged users can fetch information from the user region, page zero and their own TCB. The CREAD or VIP privilege allows you to inspect any part of the job's virtual memory.

Calling Sequence: CALL FETCH(loc,len,tobuff)

or

CALL FETCHB(loc,len,tobuff,bufpos)

Arguments:

loc	Starting addr of core to be fetched

len	Number of bytes to be fetched (any value .ge. 0)
tobuff	Receiving field
bufpos	Starting position in tobuff (.ge.1) (fetch uses bufpos=1)

FSCHEK

This routine checks the status of 3270 screen. It is usually used by full screen multi-session applications to determine if the screen can be re-written without erasing something that the user has not yet read.

Calling Sequence: CALL FSCHEK(retcd)

retcd 4-byte return code field.
- 0 Terminal is in full screen mode
- 1 MUSIC mode but no data on screen
- 2 MUSIC mode and screen contains data

PMSG

This routine sets the message flag in the TCB to allow a program to handle the messages or to suppress the automatic display of messages until the flag is reset.

Calling Sequence: CALL PMSG('op')

op ON : program will handle messages
OFF: normal message processing

PSTCOD

A call to this routine retrieves and resets the post code. If you want to look at the post code without resetting it, you can find a copy of it in location x'C64' in low core. The post code is set by the /POST command, the WAKEUP subroutine, and a variety of asynchronous system events.

Calling Sequence: CALL PSTCOD(value)

value is a 8-byte value of the post code.

QRD

Read data from a spool buffer chain managed by DSPOOL. This can be used to read the message queue, the intertask communications queue, or the terminal input and output queues.

Calling Sequence: CALL QRD(qid,tcb,buf,len)

Arguments:

qid
- 1: output
- 2: input
- 3: message
- 4: task

tcb TCB number

buf buffer address

len data length

The SYSCOM privilege is required to access queues owned by another TCB.

QRDX

Read data from a spool buffer chain managed by DSPOOL. This can be used to read the message queue, the intertask communications queue, or the terminal input and output queues.

Calling Sequence: retcod=QRDX(qid,tcb,buf,len)

Arguments:

retcod A return code, set as follows:
0 operation complete
1 failed (not enough buffer space)
2 no record found (end of file)
3 routine already processing chain

qid 1: output
2: input
3: message
4: task

tcb TCB number

buf buffer address

len data length

The SYSCOM privilege is required to access queues owned by another TCB.

QWR

Write data to a spool buffer chain managed by DSPOOL. This can be used to write to message queue, the intertask communications queue, or the terminal input and output queues.

Calling Sequence: CALL QWR(qid,tcb,buf,len)

Arguments:

qid 1: output
2: input
3: message
4: task

tcb TCB number

buf buffer address

len data length

The SYSCOM privilege is required to access queues owned by another TCB.

QWRX

Write data to a spool buffer chain managed by DSPOOL. This can be used to write to message queue, the intertask communications queue, or the terminal input and output queues.

Calling Sequence: retcod=QWRX(qid,tcb,buf,len)

Arguments:

retcod A return code, set as follows:
0 operation complete
1 failed (not enough buffer space)
2 no record found (end of file)
3 routine already processing chain

qid 1: output
2: input
3: message
4: task

tcb TCB number

buf buffer address

len data length

The SYSCOM privilege is required to access queues owned by another TCB.

ROUTE

This routine returns information from the system routing table. See $PGM:$ROUTING.S for the format of the routing table.

Calling Sequences:

CALL ROUTE(1,rtname,retcod)
or

CALL ROUTE(rtname)
This returns the default printer name for the user running the program. The routine first checks if one is specified in the user profile. It then looks for a device address match in the routing table. If these fail *retcod* is set to 1 and *rtname* is set to the default route name from the beginning of the table. In the call with only 1 argument, no return code is set, but *rtname* is set the same way.

CALL ROUTE(2,rtname,retcod)
A request to check whether the specified *rtname* is in the table.

CALL ROUTE(3,rtname,userid)
userid returns the 4-char id of the user authorized to run the printer *rtname.* If the id is

longer than 4 chars, only the first 4 are returned. *userid* is returned as 1 or 2 (like retcod) if the route name cannot be found.

CALL ROUTE(4,rtname,retcod,info,buflen)
: A request to search the table for route name *rtname*, and return information from the table entry in *info*. This *info* is the table entry except for the 1st 12 bytes. The caller sets *buflen* to the length of the *info* argument (0 or more). The bytes of the table entry after the route name are returned in *info*, as much as will fit in the buffer provided, and the remainder (if any) of the buffer is set to blanks.

CALL ROUTE(5,rtname,retcod,info,buflen)
: Similar to call type 4, except that the sequential number (1,2,...) of the table entry to be returned is put into *retcod* by the caller. The route name from the entry is returned in *rtname*, and the entire table entry is returned in *info* (12 more bytes than for call type 4, since the entire entry is returned). Table entry 1 is the default route name; table entry 2 is the first "normal" route name entry; etc. This type of call can be used to scan the table, or to get all the entries one by one.

Arguments:

rtname
: 8-character route name (e.g. a destination name for job output, or the name of an rscs remote printer). It is output (set by this routine) for call type 1 and input for the other types.

retcod
: A return code, set as follows:
  0 OK (device addr or name found in table).
  1 Device addr or name not found in table; for call type 1 (or if only 1 arg), *rtname* is set to the default route name from the beginning of the table. For call type 5, retcod=1 means the input entry number is out of range.

2
: The table could not be loaded. For call type 1, the name is set to all blanks if retcod=2. For call type 5, *retcod* is also an input argument, specifying the number (1,2,...) of the desired entry.

userid
: For call type 3, set to the first 4 chars of the user id from the table entry.

info
: User's buffer where the remainder of the table entry is returned (or, for call type 5, the entire table entry).

buflen
: The length of the area provided by the caller in the *info* argument.

Notes:

1. If the calling sequence is incorrect or call type (1st arg) is wrong, the program stops with an invalid op-code (program interrupt 1) and r8=x'EEEEEEEE'.

2. The *info* returned from the type 4 call is in the following format:

```
displ.  len.  contents
  0      8     MUSIC userid authorized to run this printer,
               or blanks.
  8      1     type of table entry:
               1 = local VM system printer
               2 = rscs printer or another virtual machine
               3 = ignore (throw away) all output
               4 = output to MUSIC's print queue
               5 = mvs printer (mcgill only)
```

```
 9      1     VM class.
10      2     (reserved)
12      8     target VM userid.
20      8     VM forms name (or blanks).
28     24     VM tag text (or blanks).
```

total length of *info* 52

(For call type 5, the above info is preceded by the first 12 bytes of the table entry, which contain the low and high address and the route name. For entry #1 i.e. the default route name, entry length and number of entries are returned instead of low and high address.)

3. If entry point "ROUTE1" is called instead of "ROUTE", then this routine finds the address of the routing table itself (by searching the LPA directory), instead of issuing the $LODSVC SVC. For example, nucleus module SPAM calls ROUTE1, since $LODSVC is not supported in supervisor state. ROUTE1 does not work outside of supervisor state because the pointer (in module LODSVC) to the LPA directory is in fetch-protected main storage. For ROUTE1, the routing table must be in the FLPA (not PLPA).

4. The subroutine is re-entrant (it does not modify itself).

SETSVC

This routine lets programs in high level languages issue the $SETOPT SVC. This SVC is used to move parameters or set options in areas of protected storage. The actual function performed depends on the argument passed to the routine.

Calling Sequence: CALL SETSVC(arg)

A number of different functions can be performed depending on the contents of the argument string. These are listed below.

A0 Set a user option bit on or off in the TCB.

```
ARG  DC    X'A0',AL1(x,y,z)
```

Where x is 0 to set bit off, 1 to set bit on. y is option byte number (1 to 4). z is bit number (0 to 7).

A1 Wake up MUSIC batch (SPAM) to process a job which has just been put into the submit data set. The SYSCOM privilege is required.

```
ARG  DC    X'A1',AL1(N)
```

Where *n* is the submit *q* number (must be 1 to 32). The *q* bit in SBMCOM is turned on, and, if *n* is the current reader class, a $CSTART SVC (reader start) is done.

A2 Force the specified terminal to be signed off. This requires the MAINT privilege.

```
ARG  DC  X'A2',AL3(TCBADR)
```

A3 Scan the ENQ table for any TCBs enqueued on a specific name and issue the equivalent of a /POST command to those programs (see WAKEUP). The calling program must have SYSCOM privilege.

```
ARG  DC  X'A3',CL8'ENQENT',CL8'POSTCMD'
```

A4 Set a user defined CVT pointer in location X'10' and a TCBOLD pointer at X'21C'.

```
ARG   DC   X'A4',AL3(USERCVT),A(USRTCB)
```

A5 Set the VIP bit on/off in the XTCB for this user this will only be done for user who are authorized for for VIP.

```
ARG   DC   X'A5','AL3(TCBADDR),X'N'
```

n - 0 set vip off
n - 1 set vip on

A6 Set the debug control block pointer.

```
ARG   DC   X'A6',AL3(BDGPTR)
```

A8 get/put from spool buffer chains. The caller must have SYSCOM to access another tasks buffer chains. (see QRD/QWR).

```
ARG   DC    X'A8',AL1(OPT),H'TCBNUM',A(BUF),A(LEN),A(PTR)
```

A9 Wakeup a task that has called delay.

```
ARG   DC    X'A9',H'TCBNUM'
```

AA Set message flag bits on or off (see PMSG).

```
ARG   DC   X'AA',H'TCBNUM',X'YY',X'ZZ'
```

yy - 80 or, 00 and
zz - data byte (only 80, 40, and 20 bits can be changed)

UTEST

Tests a userid, a userid pattern or a user type (TSUSER option 15) against a range.

Calling Sequence: CALL UTEST(string,strlen,userid,utype,rc)

Arguments:

string	a 1 to 32 character string in the form of t=n1-n2 where n1 and n2 are an integer range and/or a userid/userid pattern.
strlen	length of *string*.
userid	16 character userid.
utype	integer*4 value obtained from TSUSER option 15.
rc	is the return code. =1 hit (utype is in the range n1-n2) =0 no hit (utype is NOT in the range n1-n2) =-1 invalid integer for either n1 or n2

VMCMD

Issue a VM CP command via diagnose 8.

Calling Sequence: CALL VMCMD(cmd,lcmd,resp,lresp,retcod,outlen)

Arguments:

cmd — Command string. May contain multiple CP commands separated by x'15'.

lcmd — Length of command string, 1 to 240.

resp — Response buffer.

lresp — Length of response buffer, 1 to 8192, or 0 if response buffer should not be used. If lresp=0, VM sends the response lines to the MUSIC VM console.

retcod — Return code:

- 0 Command was issued, and executed successfully.
- >0 Command was issued but was not successful. *retcod* is a VM error number.
- <0 MUSIC error code. Command was not issued.
- -1 Interface is busy; calling program should delay for a short time and retry.
- -2 Invalid request.
- -3 Not under vm.

outlen — Set to indicate length of the response:

- >0 If the response fit in the buffer: *outlen* is the length of the response text. Response lines are separated by x'15' character.
- =0 If lresp=0 or if retcod<0.
- <0 If the response did not all fit in the buffer: outlen=-(number of bytes that would not fit). Note that this does not affect retcod.

Notes:

1. The user or the program must have the "maint" privilege.

2. High-order bit x'80' must be on in the last argument word, to indicate exactly 6 arguments.

3. The calling sequence is similar, but not identical, to that for the mug-mods-tape version of this routine, which is called "cpcmd". Return codes are a bit different, the 6th argument is added, and lresp is not modified.

4. If the SVC is busy, this routine automatically retries up to 5 times before setting retcod=-1. Therefore, the caller should not normally have to worry about testing for busy condition.

5. This routine is re-entrant.

WAKEUP

Issue $SETOPT SVC request to wake up other jobs. The other jobs are identified by an entry in the enqueue table. The SYSCOM privilege is required.

Calling Sequence: CALL wakeup(name,post)

Arguments:

name — 8-byte name. This is matched with the last 8 bytes of the 10-byte names in the enqueue table.

post — 8-byte post code to be put into the XTCB of the jobs that are woken up.

REXX Function Packages

These allow MUSIC subroutines to be called directly from REXX programs. Three function packages are supported by REXX.

```
RXSYSFN  - Provided with MUSIC/SP.
RXLOCFN  - Local function package.
RXUSERFN - User function package.
```

Only RXSYSFN is actually distributed with the system. If you want to make additional functions or subroutines available to your users, you can do so by creating RXLOCFN or RXUSERFN.

There are three major components to the function packages.

The Function Interface

The program in the file $REX:REXX.RXSYSFN.S is the interface between the standard linkage used in MUSIC routines and the parameter lists used by REXX. When a function is called this routine converts the REXX parameter list to standard linkage form, calls the subroutine or function, and moves any results back to the appropriate REXX variables.

The Function Table

The parameter conversion is done based on a table that describes the functions and subroutines that are part of the package. This table is defined by a group of PARMS macros. The file $REX:REXX.FUNCTABL.S contains the function table used for the distributed function package. Each subroutine or function and its calling sequence is defined. If a routine has variable calling sequences, each one must be defined. Comments in the macro ($REX:REXX.PARMS.M) describe how the macro is used.

The Functions

The object code for standard MUSIC functions and subroutines is linked in with the interface module and the function table to form the function package. Not all routines are eligible. For example, routines that use OS I/O access methods, have multiple return addresses, pass floating point parameters or use non-standard linkage, will not work.

Creating a Function Package

1) Create a new function table. Use $REX:REXX.FUNCTABL.S as a model. Assemble it and save the object module.

2) Link the new table with the interface module and the subroutines. The file $REX:REXX:RXSYSFN.LKED provides an example of control statements to do this. Although the interface object code ($REX:REXX.RXSYSFN.OBJ) should be used you should change the name of

the load module file and load module name to RXLOCFN or RXUSERFN, depending on which function package you are creating.

3) Put the new function package into the system load library. The file $REX:REXX.RXSYSFN.LD shows how this is done.

Segmented Function Packages

When a function is called from a REXX program the entire function package is loaded into memory. This causes overhead proportional to the size of the function package. To reduce the overhead it is possible to break a function package into segments containing related functions. Only the segment containing the required function is loaded.

The distributed function package (RXSYSFN) contains five segments.

RXSYSFN	- Standard functions (root segment)
REXITCM	- Intertask communications routines
REXLDEV	- Logical device routines
REXMFIO	- MFIO subroutines and control areas
REXSOCK	- TCP/IP Socket routines

Creating a segmented function package is slightly more complicated than an unsegmented one since each segment has to be created as a separate module.

The function table ($REX:REXX.FUNCTABL.S) contains the definitions of all functions in the package. Those that are not contained in the root segment are identified by the EXTTYP=ENTRY parameter on the PARMS macro. In addition an entry is added for each extra segment specifying EXTTYP=MEMBER and defining the name of the segment module. This function table is linked in with the RXSYSFN code to form the root segment. See the file $REX:REXX.RXSYSFN.LKED for an example.

Each extra segment consists of a stub module, created using the FHEADER macro, that defines all the routines in the segment. This is linked in with the routines that form that segment of the package. See the files $REX:REXX.REXLDEV.* for examples. The segment modules must be in the load library.

Calling Functions From REXX

To avoid problems in parameter substitution, each individual parameter must be quoted and in upper case. The parameters should be separated by commas. For example:

```
call TSUSER '4','TCBNUM'
```

Care must also be taken in how some of the routines are used. For example, when REXX schedules a MUSIC command, the system automatically closes all files and terminates any logical devices. This means that the MFIO and Logical Device routines lose control whenever the REXX program issues a MUSIC command. This problem can be avoided by using multi-tasking (NXTCMD routine), to execute MUSIC commands rather than the standard chaining technique that is used by REXX.

As well the following GT and ST take a single parameter, which should be the name of the variable that contains the BINARY version of an MFIO control block. REXX thinks that they are characters, so you must be very careful about inserting and extracting information. Typical calling sequences are:

```
call GTMFARGX 'MFARGX'     (or call GTMFARGX 'GORK' - the name is
                           arbitrary)
call STMFARGX 'MFARGX'
```

The following subroutines and available in the distributed system function package. Some routines are documented in the *MUSIC/SP User's Reference Guide*, others are documented elsewhere in this manual.

CLRIN	Clear the INPUT area on the screen.
COPY	Copy one file to another.
COUNT	Issue count SVC.
DEBUG	Call interactive debug routine.
DELAY	Stop program for a length of time.
ECHOIN	Echo input to screen.
ENQ	Call enqueue facility
EOJ	Set return code and terminate a job.
FETCH	Fetch data from storage.
FILMSG	Return file system message.
FSCHEK	Check screen status.
FSIO	Do full screen I/O.
GETRET	Get return code.
GTMFxxxx	retrieves control block information xxxx can be one of the following: ARGX, TAG, INFO, HINF, UINF, EOFP, BUPN PHYS, LIBR, UCTL, EXTN, FSAR, XINF
ITxxxx	Intertask communication subroutines (ITCOM) are described below, they include: ITSERV, ITRECV, ITSEND, ITFIND.
KEEPIN	Preserve terminal input.
LDCUR	Logical Device Interface
LDECHO	" " "
LDENT	" " "
LDGET	" " "
LDINIT	" " "
LDKEY	" " "
LDLOG	" " "
LDPUT	" " "
LDRECV	" " "
LDSEND	" " "
LDTERM	" " "
LDWAIT	" " "
LDWINT	" " "
MATCH	Pattern matching with wild card characters.
MFACT	Open/Close file dynamically.
MFGETU	Get file internal unit number.
MFIO	Do I/O to file.
MFSETU	Set file internal unit number.
NOECHO	Don't echo data to screen.
NOSHOW	Don't display input data.
NOTRIN	Don't translate terminal input.
NXTCMD	Execute multi-tasked or chained command.
PARM	Fetch parameter string.
PSTCOD	Fetch post code.
PURGE	Purge files.
QRD	Read data from a buffer chain.
QRDX	Read data from a buffer chain.
QUERY	Call query routine.
QWR	Write data to a buffer chain.
QWRX	Write data to a buffer chain.
RENAME	Rename a file.
ROUTE	Get information from routing table.

RXQFOP	Quick reading and searching of large files.
SHOWIN	Cancel call to NOSHOW
SOCKET	Socket routines are described in the section "MUSIC Socket Interface to TCP/IP" later in this chapter.
STMFxxxx	stores control block information xxxx can be one of the following: ARGX, TAG, INFO, HINF, UINF, EOFP, BUPN PHYS, LIBR, UCTL, EXTN, FSAR, XINF
SYSENV	Return information about system environment.
TRIN	Cancel call to NOTRIN
TSDATE	Get current date information.
TSTIME	Get time of day information.
TSUSER	Get user/terminal information.
VMCMD	Issue a VM command.
WAKEUP	Wakeup a task and set the post code.

ITCOM: Intertask Communication

This set of subroutines provides a simple intertask communications protocol designed for client/server applications within a MUSIC system. The SYSCOM privilege is required to use these routines.

In a typical application the server defines itself to the system using the ITSERV routine. The server name must be unique. The system reserves names beginning with "$S" for applications with the SYSCOM privilege. The server then enters its main processing loop where it calls ITRECV to wait for incoming requests from clients.

The client program locates the server application by calling ITFIND. This returns the TCB number of the server whose name is specified in the call. Once the TCB number of the server is known the client can use ITSEND and ITRECV to exchange data with the server application.

ITSEND and ITRECV are used to exchange data between tasks. Although intended to be used in conjunction with ITSERV and ITFIND in a client/server model, they can be used quite independently since they use only the TCB number to identify the other task.

ITSEND allows a program to send packets of up to 8K to another task. The data is chained in buffers off the receiving tasks TCB. An eight byte header is added to the buffer to identify the sending task. This is used internally by the subroutines and need be of no concern to the application program since it is stripped off by the ITRECV routine.

ITRECV received data from the "intertask-communications buffer chain". Application designers should note that this buffer chain is used for a number of different functions (IUCV, TCP/IP) and is reset at sign-off but not at end-of-job. ITRECV completes when data is available or the specified time out expires. If the data is from ITSEND, the TCB number of the sender is filled in, otherwise the return code is set to indicate that the TCB number is unknown. If the receiving buffer is too small as much data as possible is returned and the rest is lost.

Calling Sequences:

```
call itserv(name,rc)
```

defines a server name to the system

name	server name for enq (1-24 characters)
rc	0:name defined -1:name in use

```
call itfind(name,tcb,rc)
```

finds the TCB number of a server

name	server name for enq (1-24 characters)
tcb	tcb number of server
rc	0:server found -1:server not defined

```
call itsend(data,len,tcb,rc)
```

sends send data to a specific tcb

data	data buffer
len	length of data (1-8k)
tcb	tcb number
rc	0:data sent -1:invalid length -2:no buffers available

```
call itrecv(data,blen,rlen,tcb,time,rc)
```

sends data to a specific tcb

data	data buffer
blen	buffer length (1-8k)
rlen	length of data received
tcb	tcb number of sender
time	time to wait for data
rc	0:data received, sender's TCB number set -1:data received but sender's TCB number unknown -2:time out

MUSIC Socket Interface to TCP/IP

A number of structures and constants are used in the TCP/IP socket interface. A few of the common ones are explained below. Others are detailed in the applicable socket calls. Note that the information applies to the AF_INET domain (internet).

Network socket address or name

This is used in the calls that establish connections or send data to specific places.

```
    2      2       4              8
+------+------+----------+------------------+
| 0002 | pppp | aaaaaaaa | 0000000000000000 |
+------+------+----------+------------------+
```

0002	indicates the AF_INET (internet) domain
pppp	16 bit port number
a..a	32 bit internet address of host
0..0	binary zeros

Clientid

This is used in exchanging sockets between applications.

```
      4              8                  8              20
+----------+------------------+-----------------+-----/-----+
| 00000002 | vvvvvvvvvvvvvvvv | tttttttttttttt | 0.......0 |
+----------+------------------+-----------------+-----/-----+
```

0..2	indicates the AF_INET (internet) domain
v..v	virtual machine name
t..t	task name specified when connecting to TCP/IP
0..0	binary zeros

In the descriptions below each subroutine is known by two names. The first one in mixed case is the official name. The second name is the six character name that can be used to call the routines from languages that don't support long subroutine names like FORTRAN, ASSEMBLER, and REXX.

Accept/ACCEPT

Accept() is used by a server to complete a connection with a client. It assigns a new socket number (ns) to the connection and fills in the client's network address in the fields provided. If there are no pending connections accept() will block until one occurs unless non-blocking mode is in effect. The select() routine can be used to test for pending connections prior to issuing a blocking accept(). Once the accept() is complete the server can accept subsequent connections on the same socket (s).

```
ns  = accept(s, name, namelen)
```

ns	New socket number assigned to the connection
s	Original socket number created by the server
name	The port number and network address of the connecting client
namelen	The length of the name.

If an error occurs ns is set to a negative value. Common errors are:

```
EBADF            -9  : s is not a valid socket descriptor
EINVAL          -22  : listen() was not called
EOPNOTSUPP      -45  : s is not a stream socket
EWOULDBLOCK     -35  : non-blocking mode with no connections
                       pending
```

Bind/BIND

Bind() defines the local network address and port number that is to be associated with a specific socket.

```
rc  = bind(s, name, namelen)
```

rc	Return code

s	Original socket number created by the server
name	The port number and network address to bind to
namelen	The length of the name

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
EADDRNOTAVAIL   -49  : address is not available or invalid
EAFNOSUPPORT    -43  : address family is not supported
EADDRINUSE      -48  : address is currently in use
EINVAL          -22  : socket is already bound to an address
```

Close/CLOSE

This shuts down the socket, frees all resources associated with it and closes the TCP connection.

```
rc  = close(s)
```

Connect/CONECT

With stream sockets connect() is used by client applications to connect to servers. The server must have set up a passive open at the other end by issuing bind() and a listen() before the connect() is issued. The server should respond with an accept() to complete the connection.

In blocking mode the connect() blocks until the connection is established or an error occurs. In non-blocking mode EINPROGRESS (-36) is returned and select() must be used to determine when the connection is actually complete.

Connect() can also be used with UDP sockets to establish peer. This basically allows the caller to send datagrams using calls like read() and write() that have no room to specify a remote address. In other words connect() simply defines a default remote address.

```
rc  = connect(s, name, namelen)
```

rc	Return code
s	Socket number
name	The port number and network address of the server to connect to.
namelen	The length of the name

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
EADDRNOTAVAIL   -49  : cannot reach the remote host
EAFNOSUPPORT    -43  : address family is not supported
EALREADY        -37  : previous connection has not completed
ECONNREFUSED    -61  : remote host rejected the connection
EINPROGRESS     -36  : connection is in progress
EISCON          -56  : socket is already connected
ENETUNREACH     -51  : remote host cannot be reached
ETIMEDOUT       -60  : connection timed out
EINVAL          -22  : addrlen is incorrect
```

Fcntl/FCNTL

Fcntl() is used to control whether a socket operates in blocking or non-blocking mode. In blocking mode (the default) the application does not receive control back from the subroutine until the operation is complete. For example if a program issues a read() it is stopped (blocked) until some data actually arrives on the socket. In non- blocking mode an error (EWOULDBLOCK) is given in response to the read() if no data is present.

```
rc = fcntl(s, cmd, data)
```

rc	Return code
s	Socket number
cmd	Indicates what to do
data	Data depending on cmd

The are currently three possible calls:

```
rc = fcntl(s, 4, 4)   : put socket into non-blocking mode.
rc = fcntl(s, 4, 0)   : put socket into blocking mode.
rc = fcntl(s, 3, 0)   : rc is set to 4 if socket is in non-blocking
                        mode, and is set to 0 if the socket is in
                        blocking mode.
```

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
EINVAL          -22  : data is not a valid flag
```

GetClientId/GCLNID

This routine is used to find out the client ID by which an application is known to TCPIP. It is used in passing sockets from one application to another through the givesocket() and takesocket() routines.

```
rc  = getclientid(domain, clientid)
```

rc	Return code
domain	set to 2 for AF_INET
clientid	format is defined previously

Common errors:

```
EPFNOSUPPORT    -46  : domain in not AF_INET (2)
```

GetHostId/GHSTID

This returns the default internet address for the host running the application that calls the routine.

```
id  = gethostid()
```

id	32 bit internet address

GetHostName/GHSTNM

This returns the name of the host running the calling application. Note, this is the actual character name, not the port number/ internet address combination that are referred to in other calls as the name.

```
rc  = gethostname(name, namelen)
```

rc	Return code.
name	The returned name of the host
namelen	On input this specified the maximum size of the name buffer. On output it contains the length of the name.

GetPeerName/GPRNM

This routine retrieves the name of the remote host. The name contains the port number and internet address.

```
rc  = getpeername(s, name, namelen)
```

name	The port number and network address of the application that the caller is connected to.
namelen	The length of the name.

Common errors:

```
EBADF            -9  : s is not a valid socket
ENOTCONN        -57  : the socket is not connected
```

GetSocketName/GSCKNM

This routine retrieves the name associated with the local socket. This contains the port number and internet address.

```
rc  = getsocketname(s, name, namelen)
```

name	The port number and network address of the application that the caller is connected to.
namelen	The length of the name.

Common errors:

```
EBADF            -9  : s is not a valid socket
```

GetSockOpt/GSCKOP

This is used to get the options associated with a particular socket.

```
rc  = getsockopt(s, level, optname, optval, optlen)
```

s	Socket number.
level	Set to -1 for sockets.
optname	Integer indicating which option to get. See setsockopt() for a list of valid values.
optval	The value returned for that option.
optlen	The length of the option value.

If an error occurs rc is set to a negative value. Common errors are:

```
EBADF            -9  : s is not a valid socket descriptor
ENOPROTOOPT     -42  : optname is invalid
```

GiveSocket/GIVESK

This is used to pass a socket to another application. For a complete discussion of the givesocket() and take-socket() routines see the comments in the file $TCP:INETD.S. This shows the steps that must be taken to start up a new task and pass a socket on to it. The socket should not be closed until the other task has successfully acquired it using the takesocket() routine.

```
rc  = givesocket(s, clientid)
```

s	The socket number to be given the the other task.
clientid	The client ID of the task that you wish to give the socket to.

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
EBUSY           -16  : listen() has been called for the socket
EINVAL          -22  : clientid does not specify a valid client ID
ENOTCONN        -57  : the socket is not connected
EOPNOTSUPP      -45  : s is not a stream socket
```

IoCtl/IOCTL

Ioctl() provides access to the operating characteristics of the socket.

```
rc  = ioctl(s, cmd, data)
```

s	Socket number.
cmd	The command to perform.
data	Information passed or returned.

The following table lists the supported requests:

Name	Value (HEX)	Function
FIONBIO	8004A77E	Clear or set non-blocking mode. Data=0 to clear, 1 to set.
FIONREAD	4004A77F	Returns number of byte available for reading in data.
SIOCATMARK	4004A707	Queries if current data is out of band. Data=1 if yes, 0 if no
SIOCGIFADDR	C020A70B	Gets network interface address. Data=IFREQ
SIOCGIBRDFADDR	C020A712	Gets network interface broadcast address. Data=IFREQ
SIOCGIFCONF	C008A714	Gets network interface configuration. On input data should be set to the length

		of the output area. On output it will have an IFREQ structure for each interface.
SIOCGIFDSTADDR	C020A70F	Gets network interface destination address. Data=IFREQ
SIOCGIFFLAGS	C020A711	Gets network interface flags. Data=IFREQ
SIOCGIFNETMASK	C020A715	Gets network interface mask. Data=IFREQ

The IFREQ structure that is returned has a 16 byte request name followed by the requested information.

Listen/LISTEN

Listen() is used by a server to indicate it is ready to accept calls from clients.

```
rc  = listen(s, backlog)
```

s	Socket number.
backlog	Length of pending connection queue.

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
EOPNOTSUP       -45  : s is not a stream socket
```

Read/READ

This call reads data from a connected socket. The number of bytes read is returned in the return code (rc). In blocking mode this call will block until some data becomes available.

```
rc  = read(s, buf, len)
```

s	Socket number.
buf	Buffer to receive the data.
len	Buffer length.

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
EWOULDBLOCK     -35  : set in non-blocking mode if no data to read
```

Readv/READV

This call is similar to the read() except that the data is read into a group of buffers pointed to by an I/O vector structure. Each entry in the I/O vector contains a buffer address and a length. The buffers are filled in order.

```
rc  = readv(s, iov, iovcnt)
```

s	Socket number.
iov	I/O vector containing buffer address / length pairs

iovcnt Number of entries in the iov.

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
EWOULDBLOCK     -35  : set in non-blocking mode if no data to read
```

Recv/RECV

This receives data from a connected socket. The return code (rc) is set to the number of bytes received. It is very similar to read() except for allowing certain flags to be set.

```
rc  = recv(s, buf, len, flags)
```

s	Socket number.
buf	Buffer to receive the data.
len	Buffer length.
flags	Flag value of MSG_OOB (1) or MSG_PEEK (2)

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
EWOULDBLOCK     -35  : set in non-blocking mode if no data to read
```

RecvFrom/RECVFM

This receives data from a datagram socket. The return code (rc) is set to the number of bytes received.

```
rc  = recvfrom(s, buf, len, flags, name, namelen)
```

s	Socket number.
buf	Buffer to receive the data.
len	Buffer length.
flags	Flag value of MSG_OOB (1) or MSG_PEEK (2)
name	Port number and network address of the sending application
namelen	Length of name.

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
EWOULDBLOCK     -35  : set in non-blocking mode if no data to read
```

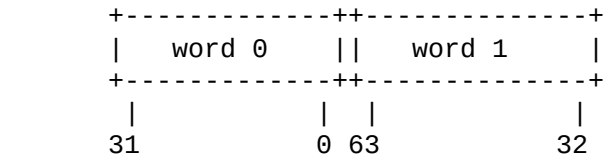
Select/SELECT

The select() routine is used to wait for an interrupt from the socket interface and to determine which socket caused the interrupt. It is used to wait for connections to come and for data to arrive. It is usually used by applications that field interrupt from a variety of sources or handle a number of different sockets and can't afford to get into a blocking situation. The select() is used to test in advance for an event before issuing the socket call to handle that event.

Select() can test a group of sockets for reading, writing, and exceptional conditions. Reading implies that data is available and a read() call will not block. Writing implies that the socket is available and is set for example when a connection comes in for a socket with a pending listen(). Exceptional conditions are events

like out of band data.

On input three bit masks are used to select which sockets should be tested. On output the return code (rc) is set to the number of sockets that have pending conditions and the bits are set in the masks to indicate which sockets they are. The bit masks are organized into 32 bit words that are counted from the low order end. It's a bit weird.




```
rc  = select(nfds,readfds,writefds,excptfds,timeout)
```

nfds	Number of sockets to test
readfds	Bit mask for read
writefds	Bit mask for write
excptfds	Bit mask for exceptional conditions
timeout	Timeout value (integer seconds, integer microseconds)

Common errors:

```
EBADF            -9  : one of the descriptor sets is invalid
EINVAL          -22  : the timeout value is invalid
EMUSINT       -2000  : a non-socket interrupt occurred
```

Send/SEND

This sends a data to a connected socket.

```
rc  = send(s, buf, len, flags)
```

s	Socket number
buf	Buffer containing the data
len	Length of the data
flags	Option flags. Valid values are MSG_OOB (1) and MSG_DONTROUTE (4)

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
ENOBUF          -55  : insufficient buffer space to send the data
```

SendTo/SENDTO

Used to send packets on a datagram socket.

```
rc  = SendTo(s, buf, len, flags, name, namelen)
```

s	Socket number
buf	buffer containing the packet
len	length of the packet
flags	Option flags. Valid option is MSG_DONTROUTE (4).
name	The port number and internet address of the application that you are sending to.

namelen The length of the name

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
EINVAL          -22  : namelen is invalid
EMSGSIZE        -40  : the message is too big to be sent as a
                       single datagram.
ENOBUFS         -55  : No buffer space available to do send
```

SetSockOpt/STSKOP

This sets various options associated with a socket. It allows you to control broadcast messages, out of band data, and tell the socket interface how to handle the close and bind operations.

```
rc  = setsockopt(s, level, optname, optval, optlen)
```

s	Socket number.
level	Protocol level (set to -1 for socket level)
optname	Option name
optval	Option value
optlen	Option length

The following table of lists the supported options. This list also applies to the getsockopt() routine. For those calls that toggle an option on or off, optval=1 sets it on and optval=0 sets it off. In both cases optlen should be set to 4.

Name	Description
SO_OOBINLINE X'00000100'	Toggle reception of out of band data. If on, out of band data is placed in the normal input queue.
SO_REUSEADDR X'00000004'	Toggle address reuse. Setting this on forces bind() to reuse an address
SO_LINGER X'00000080'	Causes interface to linger on close() if data remains to be sent. optval consists of two fullwords. The first one is a toggle. The second is the number of seconds to wait.

ShutDown/SHUTDN

This shuts down all or part of a full duplex connection.

```
rc  = shutdown(s, how)
```

s	Socket number
how	0:ends communication from socket 1:ends communication to socket 2:ends all communication with socket

Socket/SOCKET

This routine creates a socket and assigns the socket number (s).

```
s   = socket(domain, type, protocol)
```

domain	Set to 2 for the AF_INET domain (internet)
type	Type of socket.(1-stream, 2-dgram, 3-raw)
protocol	Protocol to use or 0. 0 is the default and is determined by domain and type. AF_INET and STREAM implies TCP. Valid values include:

```
ICOMP     1
TCP       6
UDP      17
Raw IP  255
```

Common errors:

```
EIBMIUCVERR    -1004  : error contacting TCP/IP through IUCV
EPROTONOTSUPPORT -35  : protocol not supported
```

TakeSocket/TAKESK

This is used by a program receiving a socket from another program. The subroutine GETSOC performs the handshaking required to get a socket from another program, including issuing the takesocket() call. For further details see the comments in $TCP:GETSOC.S and $TCP:INETD.S.

```
s   = takesocket(clientid, hisdesc)
```

clientid	The client ID of the application giving the socket.
hisdesc	The socket number used in the application giving the so

Common errors:

```
EACCESS       -13  : the other application did not give this socket
EBADF          -9  : hisdec is not a socket owned by the other pgm
EINVAL        -22  : clientid does not specify a valid client
EMFILE        -24  : socket descriptor table is full
ENOBUFS       -55  : TCP/IP has run out of socket control blocks
EPFNOSUPPORT  -46  : domain field of the clientid is not AF_INET (2)
```

Write/WRITE

This writes data to a connected socket.

```
rc  = write(s, buf, len)
```

s	Socket number
buf	Buffer containing the data
len	Length of the data

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
```

```
ENOBUFS         -55  : no buffer space for write
```

writev/WRITEV

This call is similar to the write() except that the data is written from a group of buffers pointed to by an I/O vector structure. Each entry in the I/O vector contains a buffer address and a length.

```
rc  = read(s, iov, iovcnt)
```

s	Socket number.
iov	I/O vector containing buffer address / length pairs
iovcnt	Number of entries in the iov.

Common errors:

```
EBADF            -9  : s is not a valid socket descriptor
ENOBUFS         -55  : no buffer space for write
```

MUSIC Specific Routines

The following routines are specific to the implementation of the socket interface on MUSIC/SP.

a2e/A2E

Convert ASCII to EBCDIC. Note that the default representation for character data on the internet is ASCII whereas IBM mainframes tend to use EBCDIC.

```
call a2e(buf,len)
```

buf	buffer containing ASCII data
len	number of characters in the buffer

CARGCALL

Convert socket routines calling sequences to that defined in the Berkeley sockets header file "MANIFEST.H".

```
rc = CARGCALL()
```

cnvd2x/CNVD2X

Convert a dotted decimal character format internet address to a 32 bit binary number. For example 132.206.120.2 is converted to X'84CE7802'.

```
call cnvd2x(daddr,xaddr)
```

daddr	character format dotted decimal address terminated by a
xaddr	32 bit binary internet address

cnvx2d/CNVX2D

Convert a 32 bit binary internet address to the dotted decimal character format. For example X'84CE7802' is converted to 132.206.120.2.

```
call cnvx2d(xaddr,daddr)
```

xaddr	32 bit binary internet address
daddr	16 character decimal address

e2a/E2A

Convert EBCDIC to ASCII. Note that the default representation for character data on the internet is ASCII whereas IBM mainframes tend to use EBCDIC.

```
call e2a(buf,len)
```

buf	buffer containing EBCDIC data
len	number of characters in the buffer

e2e/E2E

This removes control characters from an EBCDIC character string. Control characters are converted to blanks. Printables are left as is.

```
call e2e(buf,len)
```

buf	buffer containing EBCDIC data
len	number of characters in the buffer

Getcon/GETCON

This routine is used by a server application to wait for a connection on a specific port. It supports only one connection on the port and is intended to be used in debugging servers that would normally be run in the background by INETD. This allows the server to establish a connection without intervention of INETD and thus be debugged as a foreground task. Usually once the server has been debugged this call would be replaced by a call to getsoc() and the server would be set up to run under INETD.

```
s = getcon(port)
```

s	Socket number assigned to connection.
port	Port number

Getsoc/GETSOC

This routine used by a server application that has been scheduled by INETD to get the socket passed from INETD. It automatically handles the protocol to successfully transfer a socket from one application to another.

```
s = getsoc()
```

s	Socket number of passed socket.

Common errors:

```
EMUSBADPARM   -2001  : Parameter passed from INETD is invalid
EMUSINUSE     -2002  : TASKID is already in use
```

GtPort/GTPORT

This function gets a unique port number from the system that can be used to create unique port-number/internet-address combination that is referred to as the socket name.

```
num = gtport()
```

num unique port number between 1000-32768

Resolv/RESOLV

Call a domain name server to convert a host name into an IP address.

```
call resolv(hostname,ipaddr)
```

hostname character host domain name terminated by a blank.
ipaddr 32 bit internet address.

TrSock

This routine turns on socket level tracing (basically detailing the interface between the socket calls and IUCV). Two files are created: $TCP:@TCPIP.LOG, which contains trace records for each socket call, and $TCP:TCPIP.BUFFERS, which contains the inbound and outbound buffers at the socket level.

```
call trsock
```

Note that no parameters are needed.

Errors: none.

XPath/XPATH

This routine defines the TASKID field that is used when connecting the application to TCP/IP. This forms part of the client ID that is used in the givesocket() and takesocket() routines.

```
call xpath(taskid)
```

taskid An 8 character string that uniquely identified the task

Socket Errors

The error number is returned as a negative value as the return code of a socket call.

```
EPERM           1               Not owner
ENOENT          2               No such file or directory
ESRCH           3               No such process
```

```
EINTR           4               Interrupted system call
EIO             5               I/O error
ENXIO           6               No such device or address
E2BIG           7               Arg list too long
ENOEXEC         8               Exec format error
EBADF           9               Bad file number
ECHILD          10              No children
EAGAIN          11              No more processes
ENOMEM          12              Not enough core
EACCES          13              Permission denied
EFAULT          14              Bad address
ENOTBLK         15              Block device required
EBUSY           16              Mount device busy
EEXIST          17              File exists
EXDEV           18              Cross-device link
ENODEV          19              No such device
ENOTDIR         20              Not a directory
EISDIR          21              Is a directory
EINVAL          22              Invalid argument
ENFILE          23              File table overflow
EMFILE          24              Too many open files
ENOTTY          25              Not a typewriter
ETXTBSY         26              Text file busy
EFBIG           27              File too large
ENOSPC          28              No space left on device
ESPIPE          29              Illegal seek
EROFS           30              Read-only file system
EMLINK          31              Too many links
EPIPE           32              Broken pipe
EDOM            33              Domain error
EWOULDBLOCK     35              Operation would block
EINPROGRESS     36              Operation now in progress
EALREADY        37              Operation already in
ENOTSOCK        38              Socket operation on
EDESTADDRREQ    39              Destination address required
EMSGSIZE        40              Message too long
EPROTOTYPE      41              Protocol wrong type for
ENOPROTOOPT     42              Protocol not available
EPROTONOSUPPORT 43              Protocol not supported
ESOCKTNOSUPPORT 44              Socket type not supported
EOPNOTSUPP      45              Operation not supported on
EPFNOSUPPORT    46              Protocol family not
EAFNOSUPPORT    47              Address family not supported
EADDRINUSE      48              Address already in use
EADDRNOTAVAIL   49              Can't assign requested
ENETDOWN        50              Network is down
ENETUNREACH     51              Network is unreachable
ENETRESET       52              Network dropped connection on
ECONNABORTED    53              Software caused connection
ECONNRESET      54              Connection reset by peer
ENOBUFS         55              No buffer space available
EISCONN         56              Socket is already connected
ENOTCONN        57              Socket is not connected
ESHUTDOWN       58              Can't send after socket
ETOOMANYREFS    59              Too many references
ETIMEDOUT       60              Connection timed out
```

```
ECONNREFUSED    61              Connection refused
ELOOP           62              Too many levels of symbolic
ENAMETOOLONG    63              File name too long
EHOSTDOWN       64              Host is down
EHOSTUNREACH    65              No route to host
ENOTEMPTY       66              Directory not empty
EPROCLIM        67              Too many processes
EUSERS          68              Too many users
EDQUOT          69              Disc quota exceeded
ESTALE          70              Stale NFS file handle
EREMOTE         71              Too many levels of remote in
ENOSTR          72              Device is not a stream
ETIME           73              Timer expired
ENOSR           74              Out of streams resources
ENOMSG          75              No message of desired type
EBADMSG         76              Trying to read unreadable
EIDRM           77              Identifier removed
EDEADLK         78              Deadlock condition.
ENOLCK          79              No record locks available.
ENONET          80              Machine is not on the
ERREMOTE        81              Object is remote
ENOLINK         82              the link has been severed
EADV            83              advertise error
ESRMNT          84              srmount error
ECOMM           85              Communication error on send
EPROTO          86              Protocol error
EMULTIHOP       87              multihop attempted
EDOTDOT         88              Cross mount point
EREMCHG         89              Remote address changed
EIBMBADCALL        1000
EIBMBADPARM        1001
EIBMSOCKOUTOFRANGE 1002
EIBMSOCKINUSE      1003
EIBMIUCVERR        1004
EMUSINT            2000         Select interrupted by non-socket int
EMUSBADPARM        2001         Bad parameter
EMUSINUSE          2002         Resource is already in use
```

MUSIC IUCV Interface

MUSIC supports VM's IUCV interface. This allows communications and data exchange between MUSIC applications and other virtual machines and also between MUSIC and CP system services such as:

*BLOCKIO	- DASD Block I/O
*IDENT	- Identify
*LOGREC	- Error Recording
*MSG	- Message System
*MSGALL	- Message All System
*RPI	- Access Verification
*SIGNAL	- Signal Service
*SPL	- Spool Service

In addition IUCV can be used for communication between tasks on the same MUSIC system.

Security

In order to use IUCV, the MUSIC virtual machine must have the appropriate authorization. This involves specifying the required parameters on the IUCV control statement in the VM directory entry for MUSIC. In addition you may also want to allow more than the default number of connections by specifying the MAXCONN keyword on the OPTIONS statement. Remember that MAXCONN specifies the number of connections that can be made by the entire MUSIC machine, not the individual users. As an additional level of security, MUSIC applications must have the SYSCOM privilege to establish an IUCV connection.

Interface

The interface is essentially the same as that described in the publication *VM/ESA CP Programming Services for 370* (SC24-5435) or *VM/SP System Facilities for Programming* (SC24-5288). These manuals should be consulted to find complete information on how to write IUCV applications.

The ASSEMBLER language interface provides two macros.

The IUCV macro is used to execute IUCV functions.

```
        IUCV  function_name,options
```

Function_name The name of the specific IUCV function.

options Optional information that is function dependent.

The following functions are supported:

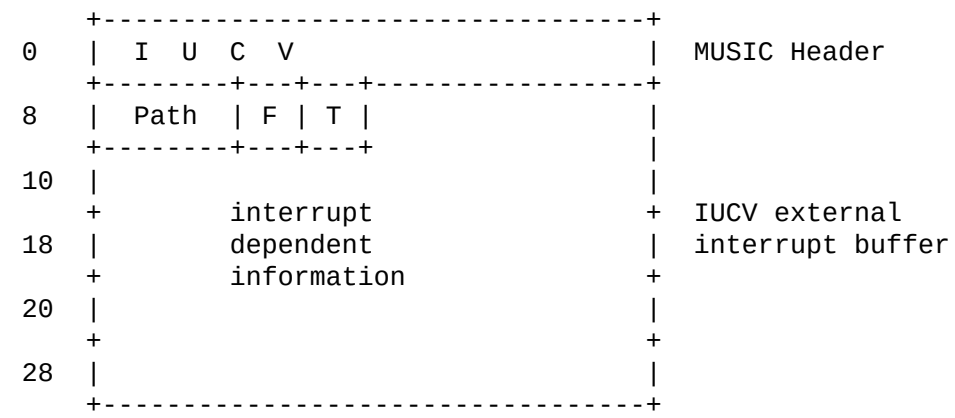
ACCEPT
CONNECT
DECLARE BUFFER*
DESCRIBE
PURGE
QUERY
QUIESCE
RECEIVE
REJECT
REPLY
RESUME
RETRIEVE BUFFER*
SEND
SET CONTROL MASK*
SET MASK*
SEVER
TEST COMPLETION
TEST MESSAGE

* These functions are supported as no-ops since MUSIC does not let applications redefine the IUCV interrupt buffer or control interrupts.

The IPARML macro used to map the IUCV parameter lists and interrupt buffers. This is copied into the assembler program.

One difference between the MUSIC and VM implementation is in the processing of interrupts. The interface module in the MUSIC nucleus gets control when an IUCV external interrupt occurs. The interrupt information is then passed to the application through the inter-task queue. The application retrieves the information

using the QRD routine. The format of this data is the same as the regular IUCV external interrupt buffer preceded by the 8 byte identifier 'IUCV '.



Path Pathid of the connection causing the interrupt

F Flags indicating the interrupt status

T Indicates the interrupt type

The IUCV external interrupt buffer is mapped by the IPARML macro.

The other difference between MUSIC and VM is how a MUSIC server application identifies itself to the system. The server program is identified by a name that it advertises in MUSIC's ENQUEUE table. A client wishing to connect to this application issues a CONNECT function specifying "MUSIC" in the IPVMID field of the connect parameter and the name of the server in the IPUSER field.

Examples

The files IUCV.SERVER and IUCV.CLIENT provide ASSEMBLE language examples of IUCV applications. In addition to showing the use of the IUCV macros, they show how to user the DELAY and QRD routines to handle the interrupts. The SYSCOM privilege is required to run these programs.

Internals

The interface handles the IUCV requests from MUSIC applications, keeps track of what application owns a conversation and passes the external interrupts to the appropriate application.

During system initialization, a DECLARE_BUFFER is issued to define MUSIC's IUCV external interrupt buffer. There is only one of these for the MUSIC virtual machine. When a task issues a DECLARE_BUFFER to initiate IUCV, it is basically treated as a NOP operation. When an external interrupt arrives that applies to that task, the interrupt handler passes the information to task via the inter-task communications queue. This information can be retrieved using the QRD routine. This handling of the external interrupt buffer data makes the MUSIC implementation a bit different from CMS.

When a task issues a CONNECT function the assigned PATHID is saved with the TCB number the PATHID_TABLE. When an external interrupt occurs, this table is used to find the TCB number of MUSIC task that is to receive the interrupt information.

If an outside task is to initiate a connection with a MUSIC task it must know the resource name associated with the MUSIC task. The MUSIC task sets the resource name in the ENQ table. So when an interrupt

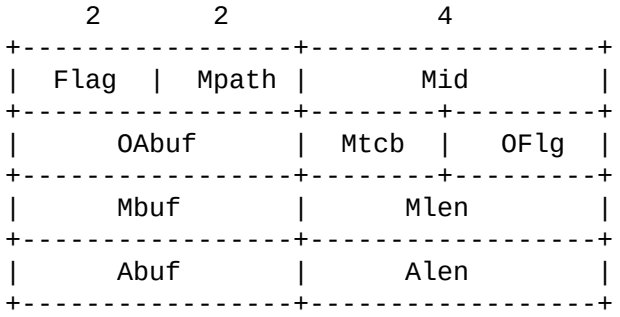
comes in requesting a connection, the TCB number of MUSIC task is retrieved from the ENQ_TABLE.

In addition to executing the IUCV instructions and passing on interrupts this module must also deal with MUSIC's virtual storage. The request parameters are copied from the user region to system storage before the request is made. The results are copied back afterwards. If a request involves data buffers, these are allocated in the system area and the virtual addresses replaced by real addresses. Once the request is complete, the data is copied to the user's virtual storage.

Message Control Blocks (MCB)

These are used to keep track of outbound messages. When a SEND or REPLY is issued an MCB is allocated. Information about the message, such as the buffer pointers, length and message ID are saved in the MCB. When IUCV finishes processing the message, it causes an IUCV external interrupt. The message ID field from the interrupt is used to locate the MCB. The MCB and any allocated buffers are then freed.

They are mapped by the MCB Dsect and are laid out as follows..



ABUF and MBUF point to buffer chains. The first buffer is used to contain a buffer list in the format used by IUCV. Subsequent buffers contain data. This allows a 32K message to span up to 64 buffers.

When an IUCV external interrupt occurs, URMON passes control to the IUCV external interrupt routine in the module IUCV. The function of this routine is to pass the external interrupt and any associated data to the appropriate MUSIC task. Since the tasks address space may be inaccessible to us, the interrupt buffer data is passed via the inter-task communications queue.

If the interrupt is for a connection pending, the ENQ table is scanned for a match on the resource_id from the interrupt buffer. If a match is found the interrupt information is passed to the task that enqueued on the name and an entry is made in the PATHID TABLE to permanently associate the PATHID with that task.

Otherwise the PATHID table is used to find out the TCB number of the task. Once the TCB number is known the interrupt buffer is sent to it via the inter-task communications queue.

Chapter 23. Performance and Tuning

Overview

This section discusses how you can improve the performance of your MUSIC/SP system. The basic steps involve observing the performance of your system and making changes and seeing the results. MUSIC comes with a number of performance tools that can help you measure your system's performance. It is strongly recommended that you regularly monitor your system's performance and not just at times of poor performance. This will allow you to see the compare the times of poor performance to those of good performance and see what has changed.

Basic Resources

There are 3 basic resources that are available to MUSIC. They are:

- Processor cycles. When MUSIC is not run under VM, then this is simply the speed of your processor. Under VM, it also refers to the amount of time available that MUSIC will get from VM. There are several performance options under VM that relate to how much time MUSIC will get.
- Main storage. This is the amount of main storage that MUSIC can use. Under VM, this is its "virtual machine size".
- Disk. This includes the number of channel paths available to the disks as well as the number of disks.

To tune the system it is important to see how MUSIC is using these resources and to determine which one can be improved and at what cost.

VM Performance Considerations

Refer to *Chapter 2 - Running MUSIC/SP Under VM* for very important information on how to best configure MUSIC under VM. In particular note that you must always run a production system in either a V=R or V=V locked environment. Just locking part of MUSIC's pages WILL NOT WORK correctly. The reason for this is that VM will stop the entire MUSIC system on ANY page exception within MUSIC and not process ANY work for MUSIC until the page exception has been resolved by VM reading in the page from disk. This can take 50 milliseconds or longer on a busy machine and will cause very uneven performance. Remember MUSIC is supporting many users in the one virtual machine and not just a single user as in the case of CMS. It is therefore quite reasonable to give MUSIC much better performance options that you give to a single CMS user.

Note that running MUSIC in a V=R or locked page environment will not give MUSIC preferential treatment over your other virtual machines. What it will do is save cycles that VM would otherwise have to use to resolve page exceptions on behalf of MUSIC. In the V=R case, VM will also save cycles by not having to translate MUSIC's I/O channel programs. These cycles saved are a benefit to ALL your virtual machines. This is because it gives effectively makes the processor faster. Running MUSIC V=R can sometimes buy back 10 or 20% of your processor cycles.

Dynamic View Your System's Performance

The SSTAT utility is valuable for seeing what is happening to your system now. It can be run by typing SSTAT as a command or from the ADMIN facility's "Information and Statistical" menu and selecting "Display System Status". Its display is updated several times per minute. You should run at different times in the day to view of the load before making any changes based on its output.

The following notes point out key items of interest in performance and tuning work.

- The storage size gives the amount of storage MUSIC has available to it.
- The MAXRRS shows how much storage each user region can get to run applications. The MAXMPL shows how many regions can be active at one time.
- The number of users show how many users are currently signed on. The number of sessions show how many sessions are active. A single user can have several sessions active at one.
- The response time is the response time this program is seeing. It is an indication of the system response time but is not an average response time people are getting. (The response time people will see depends on the queue the system has placed them in. This depends in turn on what type of work they are doing.)
- The CPU usage gives the % of the CPU MUSIC is using. This includes any VM overhead on behalf of MUSIC.
- The "running" number shows the number of tasks that are running. Many of them may be waiting for input from users. That number is shown in the "idle" field. The number actually running in the user regions is shown in the "active" field. The number that is waiting to get into a user region is shown in the "queued" field.
- The "in core" field shows how many of the running tasks are in main storage. The "swapped" field shows how many tasks the system had to swap out to disk.
- The table in the display shows the total number of specific events that have occurred since your system was last IPLed. It also shows the rate of these events. The rate is the more interesting number.
- The number of I/O operations per second is an indication of the how busy your I/O devices are. If you see this number never gets higher than a certain number at peak times, then this could mean that you have an I/O bottleneck. The IOTIME utility can be used to analyze your I/O activity.
- The number of page events per second is an indication of how much paging you are doing. It does not directly show your paging rate in terms of number of pages read or written per second as explained below.
- The "page wr" figure shows the number of page write operations being done per second. Each of these operations typically writes 8 4K pages to disk in one disk I/O operation.
- The "page rd" gives the number of page reads done. Page reads are done one at a time. Page reads done from the PLPA area are shown separately. Large paging rates here are an indication that some users are running jobs do not fit well in the MAXRRS area. You can increase the size of the MAXRRS area by doing a NUCGEN but that may increase the amount of swapping done.
- PLPA pages are for routines that have been placed in the PLPA area. They are typically reentrant compilers. PLPA page reads are done one at a time. A high PLPA rate would be an indication that you should move some items into the fixed link pack area. Consult the section "Link Pack Area Considerations" in *Chapter 21 - Load Library and Link Pack Area* for information on this topic.

- The number of swap reads and writes shows the rate of swapping. Each swap will typically take several I/O operations. This is because each swap piece is 40K in size. Refer to section "Paging and Swapping" in *Chapter 18 - System Internals* for more information.

Main Storage Usage

Many sites have found that they are not using main storage in an optimal fashion. The output from the MAPMEM utility shows you how your main storage is used. It can be run by typing MAPMEM as a command or from the ADMIN facility's "Information and Statistical" menu and selecting "Display Memory Map". The numbers displayed will not change until some system parameters are changed and the system re-IPLed.

The following notes point out key items of interest in performance and tuning work.

- The first information listed is similar to that shown with the SSTAT utility described above. Take note of the page pool size shown on the first line. That is the amount of storage available for user regions. User regions are the area that user jobs run in. A specific user job cannot use more any more real storage than shown in the MAXRRS number. The size of the page pool is determined by what is left over when all the other storage areas have been allocated.

- The length column in the table of addresses is the most important field for performance and tuning considerations. The amount used by the BPOOL area is important. Run the BPOOL utility at the end of a typical day to see the maximum number used. (The system must have been up during your peak times to have useful numbers here.) You can change the number of buffers in this pool by a parameter in the NUCGEN utility. Refer to the topic "Terminal Buffer Pool" in *Chapter 18 - System Internals* for more information.

- The LPA area shows the amount of storage being used by your fixed link pack area. Increasing this area reduces the amount of size of the page pool. Consult the section "Link Pack Area Considerations" in *Chapter 21 - Load Library and Link Pack Area* for information on this topic.

- The RAMDSK area shows the memory reserved for use as a RAM disk. Files are loaded into here when the system starts up and can be subsequently referenced without any physical I/O. Usually significant performance benefits can be gained by reserving about 400K of memory and loading in frequently used files. See the section "Configuring your RAM Disk" for more details.

- Cache all the Save Library space bit maps in memory, by specifying ULMAPS=1 in the NUCGEN utility.

- The trace area is set by a parameter in the NUCGEN utility and is usually no more than 20k.

- The nucleus area and low core areas are used by the system for its control code and buffers. These areas are not changed for tuning reasons.

- Ignore the page pool 1 and 2 numbers for tuning purposes. Use the page pool total on the first line instead.

Tuning Examples

The following are suggestions of what to do to improve specific performance bottlenecks.

Poor Response, CPU Usage High (eg 90%)

The demand for CPU cycles is greater than can be provided by your CPU. If this situation exists considerately throughout the day, you should consider reducing the number of users or upgrading your CPU. If it occurs periodically it may be caused by a group of users (like a student lab) running a specific piece of software. Have them use that software during non-prime time or spread its use out over the day rather than having them all run it at once. Maybe change from a lab format to a "work on your own" format.

Poor Response, CPU=40%, Swapping High

In this case the system may be I/O bound on the SWAP data sets. The output from the IOTIME program will indicate the amount of time spent swapping. There are various things that can be done to improve this. They are listed below in the order that we feel will have most effect.

- Give the MUSIC machine more main storage. This will enable the system to keep more users in storage and therefore reduce the swap load.

- Spread the swap load over more than one channel if possible. It is best to place the swap data sets on disks that have very little other activity on them. These disks should be on channels used exclusively by MUSIC. Some sites have found that some improvement can be obtained if the second (or third) SWAP data set is placed on a "low usage" VM channel. If this is attempted monitor the SWAP times very closely since if the channel becomes busy, the "solution" could be worse than the original problem.

- If you have specified a large MAXRRS value in the NUCGEN you may want to try reducing it. If a lot of people are using large region sizes making MAXRRS smaller can significantly reduce the amount of data that is swapped. This will increase the paging overhead for those using large user regions but will improve system performance for the majority of users.

- Optimize the location of the SWAP data sets if possible.
  - MUSIC owned channel (no interference from VM)
  - Dedicated disk (not minidisk)
  - Low usage pack if possible
  - Near center of pack

Poor Response, CPU=5%, I/O Rate Low

This is probably a problem in the I/O configuration or VM environment. MUSIC is spending most of its time in WAIT state. The key is to figure out what it is waiting for. Usually it is either waiting for an I/O interrupt from a terminal or a disk, or an external interrupt from a system timer. Whatever the case this situation should be reported as a system problem.

Response Ok Except for a Few Users

Look at the paging rates. It will usually indicate a significant amount of paging activity. The jobs that take a long time probably use a large user region and have a working set (of pages) larger than the MAXRRS. They are "thrashing". That is, spending the majority of time paging and doing very little useful work. Jobs like this do not seriously effect the performance of the rest of the system, however it can really be frustrating for the user since they have no idea of why their job is taking so long due to the exponential nature of the effect.

e.g. Waterloo Script Job (MAXRRS=232K)

100 page document.....5 minutes
300 page document.....2 hours
600 page document.....Cancelled after 18 hours.

There are two possible solutions.....

- If the thrashing program is reentrant then placing it in the FLPA will reduce the user region storage requirements by the size of the program and thus increase the available working storage.

- Increase the MAXRRS so that it is larger than the working set of the program. The value of MAXRRS effects other aspects of system performance. These should be monitored to determine whether the gains obtained by increasing MAXRRS are not outweighed by loses in other areas.

    - Increasing MAXRRS will reduce the number of tasks that can be kept in the page pool thus potentially increase the swap load. If you have sufficient main storage this should not be a problem.

    - If you have insufficient storage increasing MAXRRS will reduce the MAXMPL (maximum multiprogramming level). Usually a MAXMPL of 3 or 4 is sufficient.

    - The MAXRRS value indicates the maximum data transfer amount on a SWAP operation. If a lot of users are using large region sizes, increasing MAXRRS will increase the data transfer overhead for each swap operation. This may be offset however by substantial savings in paging overhead.

Poor Response at Peak Times

Performance is ok during most of the day except for the peak times. These peaks may last for half an hour or more. The response time monitor will show that the user region is active 100% of the time during these periods. CPU and I/O rates are not excessive and are similar to the ones you normally got from the machine at peak times before response started getting poor. The current situation is that the number of users (or the jobs they are running) have exceeded the capacity of your CPU current configuration. There is probably no obvious bottleneck. You must look at output of the performance measuring tools in more detail. The following is a list of things that you can try. The list is in order of what we feel will give you the most improvement.

- Give MUSIC as much main storage as possible and optimize its use as follows:

    - Place any high usage processors in the FLPA. Note the usage of processors such as VS/FORTRAN, and VS/PASCAL may vary depending on the time of year at academic sites. If you cannot fit them all in the FLPA, at least put the most used ones there. If you do not have any high usage reentrant processors, do not waste the storage by putting infrequently used modules in the FLPA. Use the output of the LDCNTS program to see which modules are being used the most. Use the LDLIST program to see the sizes of the individual modules. Refer to the topic "Link Pack Considerations" in *Chapter 21 - Load Library and Link Pack Area* for details on what processor each module is associated with.

    - If you have not already done so, reserve about 400K of memory for use as a RAM disk and load any frequently used files into it. Use the RAMREP utility to determine if the files loaded in RAM are being used frequently enough and replace any low usage files with others that you feel may have a higher hit count. A properly configured RAM disk should be able to handle 15% - 25% of file opens.

    - The number of RCBs effects the swap load. In adding main storage you should hope to increase the number of RCBs. Try to keep the number of RCBs to at least 25% of the number of active users. Once the key modules have been placed in the FLPA, additional storage is best spent in

additional RCBs. The number of RCBs is automatically determined from the size of the pageable storage pool. This pool contains the storage that is not allocated to other things like the FLPA, BPOOL, trace table, etc.

- Try to keep the MAXRRS to at least 232K. The system will reduce this value if there is insufficient storage left due to trying to put too much in the FLPA.

- If storage is at a premium remove any terminals, BTRMs and extra sessions (XSES) that are not required. Set the size of the BPOOL to be slightly higher than the high water mark indicated by the BPOOL program. Never let the BPOOL size be considerately equal to this high water mark. Reduce the trace table size (MAXTRC) to 8K.

- Improve MUSIC's virtual machine environment where possible.
  - Run MUSIC V=R or lock ALL its pages
  - Use dedicated disks not minidisks
  - If disks are on separate real channels, configure than as such on MUSIC.
  - Put all your MUSIC disks on different virtual channels. For example, if you disks are addresses 130, 131, 132, 134, try defining them to VM as 130, 230, 330, 430. This allows VM to better service the disks.
  - Do not use VM's multi-path support to access disks. This has shown to have a major negative effect on throughput.

- Improve MUSIC's disk configuration by adding channels or disks or moving high access data sets to channels or disks that are not so busy. If multiple real channels are used the swap and page load should be spread over all the channels. High access data sets such as SYS1.MUSIC.UIDX should be placed in the center of low activity disks on low activity channels. The IOTIME utility is useful in determining I/O activity on a data set by data set basis.

- Force a reduction in the user load. This is usually the least desirable though often the only resort once all of the above has been tried. There are two basic methods:

  - Restrict the usage of certain software to certain users or certain times. Subset environments can be set up using TMENU or REXX to restrict users to the assigned task and eliminate game playing or private projects which can consume resources.

  - Limit the number of users by limiting the number of terminals defined on the system. This may seem cruel but, when the peak time of the year rolls around and you have tried all of the above and response is still poor, it is probably better to have 100 people getting work done and 20 waiting for a terminal than to have 120 sitting at terminals getting work done slowly.