Jacquard Systems

SYSTEM II

Reference Manual

V1-005

SYSTEM II

REFERENCE MANUAL

Publication V1-005-13
October 1981

REVISION HISTORY

Number	Date	Notes
V1-005-0	3/75	Initial release.
V1-005-1	7/75	Replacement.
V1-005-2	9/75	Revision.
V1-005-3	12/75	Replacement. Release 2 of System II.
V1-005-4	2/76	Revision.
V1-005-5	3/76	Revision.
V1-005-6	8/76	Replacement. Release 4.
V1-005-7	11/76	Revision. Release 5.
V1-005-8	5/77	Replacement. Release 6.0.
V1-005-9	1/78	Revision. Release 6.1.
V1-005-10	12/78	Replacement. Release 7.0.
V1-005-11	3/79	Revision. Pages replaced: Title, ii, TOC-5, 10-3 to 10-7, A-1 to A-4.
V1-005-12	5/79	Revision. Release 8.0, including J500. Pages replaced: Title, 0-2 (was ii), 0-3 (was iii), Table of Contents, 1-1, 1-2, 2-3 to 2-6, 3-1, 3-2, 9-1, 9-2, 9-26 to 9-36, Section 10, 11-4, 11-17, 12-1 to 12-5, Section 15, B-9, Index.
V1-005-13	10/81	Replacement. Release 9.0.

RELATED DOCUMENTS

Number	Title
V2-005	System II Utilities Manual
V3-005	System II Error Messages

PREFACE

The information in this manual falls into two general categories:

1. The structure and function of System II, the standard operating system software for AM Jacquard Systems computers.

2. Interfaces which provide System II services to assembly language programs.

For application programmers who use Super BASIC or the Data-Rite package, some information in the first category may be useful as a supplement to the System II Utilities Manual (V2-005), the System II Error Messages Manual (V3-005), and the language manuals. In particular, the following material is suggested:

Chapters 1, 2, 3, and 5.

Chapter 6: page 6-1 to 6-6.

Chapter 7: pages 7-1 to 7-8.

Chapter 8: pages 8-1 to 8-5.

Chapter 9: pages 9-1 to 9-6; 9-26 to 9-35.

Chapters 10, 11, 12, and 15: as relevant.

(THIS PAGE INTENTIONALLY BLANK)

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

(THIS PAGE INTENTIONALLY BLANK)

Chapter 1

INTRODUCTION

System II, the operating system for Jacquard Systems computers, performs scheduling, resource management, and file management for multiple users executing a mix of identical or unrelated programs. For programs written in Super BASIC, Data-Rite, or assembly language, the available services include:

- Device and file I/O routines
- Terminal screen formatting routines
- Terminal keyboard editing and programmable function keys
- Memory management routines
- Sort routines
- Communication protocols

TERMINALS

System II is a terminal-oriented system. Any terminal can function either as an operator console or as an I/O device available to user programs. Terminal services include preliminary editing of keyboard input according to the current mode of the terminal, and extended capabilities via user-programmable function keys.

The terminal may be in control mode or in user mode. In control mode, the operating system interprets input. In user mode, the operating system passes input to the user program; the screen can be formatted for field-oriented input, or it can be used as a line-by-line device similar to a typewriter.

COMMANDS

Operators interface to System II through commands entered at any of the terminals connected to the system.

A command consists of the name of the disk where the program resides and the file name of the executable program, optionally followed by parameters to be interpreted by that program. System II responds to a command by loading and executing the requested program (or, when appropriate, by re-entering a copy already in memory). During execution, the program remains in control of the terminal unless that device is deliberately released.

RESOURCES

Memory is divided into an area for the resident portion of the operating system, a number of user partitions for program execution, a buffer pool area, and (for certain hardware configurations) a time-sliced display area.

Application programs interface with the operating system through service calls. These calls provide for I/O to all system-supported devices, resource management, task scheduling, file management, terminal I/O, and communications. When a program terminates, its resources are recovered automatically by System II.

Chapter 2

SYSTEM II STRUCTURE AND FILE CONCEPTS

FILE INPUT/OUTPUT CONCEPTS

System II provides support for both sequential and hash files. A sequential file can be viewed as a string of 8-bit characters whose ordering is arbitrary but fixed at the time the file is written. File I/O facilities in the system allow data from sequential files to be transferred in records whose size is either variable or fixed.

A hash file consists of discrete records, each of which has an identifying key field. The address of a record is found by applying a randomizing function to its key value. Access to a particular record is therefore independent of other records in the file.

File size is subject only to the restriction that System II does not provide for files which span device boundaries. That is, every file must reside on a single storage device.

The system supports a variety of devices, including moving-head cartridge disk drives, flexible (foppy) disk drives, CDC storage moudle disk drives, magnetic tape drives, line printers, character printers, communication lines, and computer terminals. Magnetic tape drives support only sequential files; disk drives support both sequential and hash files.

Disks must be formatted before they can be used by System II. The formatting process writes the required logical structure onto the disk. The structure includes a file name directory, disk space allocation records, a bootstrap area, and the data sector pool. The file name directory is used to keep track of the name, attributes, size, and location of each file on the disk. The disk space allocation records indicate which data sectors are in use and which are available. The bootstrap area is reserved for a bootstrap program which is loaded by the Automatic Program Load (APL) facility.

FILE ATTRIBUTES

In early EDP practice, a sharp distinction was drawn between programs and files of data. A payroll application program, for example, was seen as having very little in common with a file of weekly time cards processed by the program. In current data management thinking, however, the distinction between programs and files has virtually disappeared, as it serves no useful function. For purposes of storage or transfer, it is simpler to consider all programs and named sets of data records as files.

That unifying concept is carried a step further in the I/O management facilities of System II, which view (1) writing a record out to a data file and (2) writing that record out to a printer as much the same event. In that view, nondisk physical devices become files.

All files in the system have attributes that describe their characteristics. In the case of nondisk devices, these attributes are fixed, defining such operating characteristics as "read only" or "write only." These files are intrinsically "attribute protected;" their attributes are fixed and cannot be changed.

Files residing on disks are assigned attributes which reflect usage of the data they contain: read only, write only, secure, permanent, execution sharable, etc. An exhaustive list of attributes is given in Chapters 7 and 8, which discuss sequential file and hash file management, respectively.

Supporting multiple users requires the system to arbitrate conflicting claims for files. The ability to provide file sharing depends upon interpreting the attributes assigned to files. If a particular file has been declared "read sharable," for example, the system is able to maintain the file open for reading by several independent users. Similarly, if a file has been given the "write sharable" attribute, multiple users are permitted to write into the file concurrently.

The characteristics of being execution sharable and top loading are relevant only for binary files executable under the system. Program files which are declared execution sharable must be reentrant and capable of supporting multiple users. The system job initiator keeps track of the execution sharability of the programs in all active partitions. Top loading programs are loaded into the uppermost available partitions of memory while other programs are loaded into the lowest available partitions. Chapter 6 describes the System II segmentation of user-available memory into partitions and the strategy for their allocation.

FILE REFERENCING CONVENTIONS

A file is referenced in a command in accordance with one of the following formats:

```
device
device:filename
device:filename.extension
```

The name to be written in place of "device" consists of up to six letters and numerals, the first of which must be a letter. Exactly the same rule applies to "filename." An extension (a way of giving similar but not identical names to closely related files) consists of either one or two characters, chosen from any combination of letters and numerals.

Examples of perihperal device names include LPT1 (line printer) and DE02 (double-density cartridge disk). Within a command, a reference to a perihperal device omits any filename.extension suffix:

```
DSKOPY DE01 DE03
ACUP/C LPT1 LU
```

The first command causes the utility DSKOPY to copy the contents from device DE01 to device DE03. The second command causes the ACUP utility to change the characteristics of line printer LPT1 so that it will print lowercase as well as uppercase letters.

Where a command references a particular file, the file is identified by either the second or third of the formats given above: a specific file name, perhaps with an extension, following the name of the device on which the file resides. For example:

```
DELETE FP00:INFO.AP
```

However, a file residing on either the "primary" or the "nominal" disk (explained below) may be referred to without explicitly naming the device.

Note that a period - and no other character - is required to separate a file name from an extension. The colon separating device and filename may be replaced with a semicolon, as in these examples:

```
DELETE FP00;INFO
DP01;J1639.RB
```

The commands in all of the foregoing examples, like every System II command, are file names that conform to the stated conventions: that is, a command is itself the name of an executable file, which must be a single module as output by the Assembler, Super BASIC compiler, Data-Rite compiler, Report-Rite compiler, or the Relocating Link Loader (RLDR).

PRIMARY DISK

For all the terminals attached to a computer, a particular disk is designated the "primary" disk. Typically, software programs are stored on the primary disk. Files on it can be referenced from any terminal by writing a "$" character in place of the device name (followed by a colon). For example, if diskette FP01 is the current primary, then the following references are equivalent in a command input from any of the attached terminals:

```
FP01:GTAD
$GTAD
```

The CHPRI utility changes the primary or displays its name depending on the format used to call it. Changing the primary via CHPRI from any terminal changes it for all the terminals attached to the same computer.

NOMINAL DISK

While all terminals share the same primary, each can have a unique "nominal" disk. Typically, word processing documents or data files are stored on the nominal disk. The file on the nominal disk of a particular CRT can be referenced without specifying a device; that is, in the absence of a device name or dollar sign, the default is the nominal disk. For example, if the nominal for CRT1 is FP01, the following two references are equivalent:

```
FP01:AFILE
AFILE
```

The CHNOM utility changes the nominal or displays its name, as needed. A change affects only the terminal from which CHNOM was called.

DEVICE NAMES

Listed below are System II's standard device names. They are reserved for the indicated purpose and must not be used as disk file names. In each category, additional names can be formed by incrementing the suffixed decimal number, e.g., ACY2, ACY3, etc.

- ACY1 Asynchronous remote terminal
- ASY1 Asynchronous communications line
- BD00 Storage Module disk
- BMA1 BSC line, multipoint, ASCII control
- BME1 BSC line, multipoint, EBCDIC control
- BPA1 BSC line, point-to-point, ASCII control

- BPE1 BSC line, point-to-point, EBCDIC control
- COM1 General synchronous communications line
- CRT1 J100/J105/J50/J500 computer terminal
- DA00 Cartridge disk, double-density, 512-word sectors
- DD00 Cartridge disk, double-density, 9144 256-word sectors
- DE00 Cartridge disk, double-density, 9744 256-word sectors
- DH04 Multiplexer line
- DIL1 Auto-dialer
- DP00 Cartridge disk, single-density
- DPL1 Diablo printer (or compatible Qume or NEC), legal-size page
- DPR1 Diablo printer (or compatible Qume or NEC), letter-size page
- FA00 Floppy disk, double-density, double-sided
- FD00 Floppy disk, double-density, single-sided
- FP00 Floppy disk, single-density, single-sided
- LPT1 Line printer
- MCA0 Multiplexed remote terminal
- MT00 Magnetic tape
- MXA0 Multiplexer line
- MXB0 Multiplexer line
- NPP1 NEC printer, proportional spacing, PS wheel
- NPR1 NEC printer, non-proportional spacing, non-PS wheel
- NPS1 NEC printer, with sheet feeder
- QPR1 Qume printer, standard
- QPS1 Qume printer, with sheet feeder
- QPW1 Qume printer, WideTrack
- QTR1 Qume printer, TwinTrack
- UJE1 UNIVAC RJE line

(THIS PAGE INTENTIONALLY BLANK)

Chapter 3

SYSTEM FILES AND COMMANDS

SPECIAL FILES

Certain disk files have reserved names, typically with the extension "SB" for "System Binary." When such files are required, they must be available on the current primary disk. These files include: compiler language run-time libraries; non-resident device drivers; command processors for the Type-Rite word processing package; overlays for SGBASC, one of the Super BASIC compilers; and bootstrap programs which the FORMAT utility copies onto new disks.

A complete list of these files may be found in a separate publication, the Software Guide (V1-073). The following examples are selected for their general interest:

- DPRDRV.SB - Character printer driver
- CDODRV.SB - Line printer driver
- CHRGEN.SB - Default dot patterns for J500 screen
- DEBUGX.SB - Debugger
- DEINTR.SB - Data-Rite run-time library
- FOFP1.SB - Bootstrap program, FPnn on J100
- FOFPU.SB - Bootstrap program, FPnn on J500
- SRTLIB.SB - Super BASIC run-time library
- SORTM.SB - Generalized sort module

SYSTEM OVERLAYS

For certain configurations of System II, some of the system code is in overlays which are kept on the primary disk. These overlays are read into memory and executed in a special overlay area immediately following the resident portion of the system. Several overlay configurations are available, which provide a variety of trade-offs between resident system size and overlay loading time. Consult the J100 or J500 System Generation manuals for details.

A note of warning: since the system always loads its overlays from the primary disk, the user needs to be especially careful whenever that disk must be removed or switched.

The primary disk can be removed safely (after entering a REMOVE command) or switched to another device safely (after entering a CHPRI command) only if the system configuration has no overlays, or if the new disk contains an identically configured version at the same software release level.

COMMAND LANGUAGE

Unlike many operating systems of comparable power, System II does not limit terminal facilities to a fixed set of commands. Instead, each system command is, in fact, simply an execution request for a program.

As a result, user application programs can be designed to supplement, or even to replace, the standard general-purpose programs distributed by AM Jacquard.

Command line parsing, at the assembly language level, is described in Appendix B of this manual.

For details on standard programs and on the syntax of their execution requests, refer to publication V2-005, System II Utilities.

Chapter 4

LINKAGE FACILITIES

SYSTEM CALLS

Transfer of control to System II from an assembly language application program is accomplished by executing a JSR indirect routine (Jump-Save-Return to Address) through location 0003. The standard procedure used on system calls is to handle the JSR @3 mnemonically with the assembler directive .FORM (refer to System II Assembler Manual, V1-024):

```
.FORM CALL,16(X'2C03)
```

A statement containing CALL in the operation field can then be used to transfer to the system when necessary. One such .FORM directive is required in each assembly module where CALL statements appear.

Immediately following each CALL must be a word containing a constant that specifies the system entry point desired. The convention established for this purpose is to use the .WORD pseudo-operator with the name of the entry point in the operand field.

All system entry points used within a module must be declared as external references in .GLOBL statements; absolute values are assigned when that module is link-edited (refer to System II Utilities manual, V2-005, RLDR) with a special module called SYSSYM.RB. (SYSSYM.RB assigns values to the global names of all system entry points. It is furnished as part of the standard System II software. Refer to Chapter 14 for more information.)

LINKAGE PARAMETERS

Input and output parameters on system calls are contained in registers. Except where noted, a register not used for parameters is restored to its original value. However, the contents of the status register are destroyed, and SEL (the Select flag, used in "Branch on Condition" instructions) is always off after a system call.

If an error return is taken, AC0 will contain a code identifying the type of error that occurred. Each possible error code is assigned a global symbolic name and a value in a special module called ERCODE.RB. These symbolic names are listed in the Error Messages Manual, V3-005.

If a user program wishes to examine AC0 to determine which type of error occurred after an error return is taken, he must link-edit ERCODE.RB with his program and declare all error code symbols used as global. (ERCODE.RB is also provided as part of the standard System II software.) One possible way to determine which type of error occurred is to use the SKNE instruction on AC0 as illustrated below:

```
        .GLOBL  ERRA,ERRB,PARM ;EXTERNAL REFERENCES
        .FORM   CALL,16(X'2C03);DEFINES THE "CALL" OPERATOR
           .
           .
           .
        CALL                   ;SYSTEM LINKAGE
        .WORD   PARM           ;PARAMETER FOR SYSTEM ENTRY
        JMP     ERTN           ;ERROR RETURN
                               ;NORMAL RETURN
           .
           .
           .
ERTN:   SKNE    0,A            ;SKIP NEXT INSTRUCTION IF (AC0) ≠ (A)
        JMP     LOCA
        SKNE    0,B            ;SKIP NEXT INSTRUCTION IF (AC0) ≠ (B)
        JMP     LOCB
           .
           .
           .
A:      .WORD   ERRA           ;(A) = HEX VALUE, ERROR CODE "ERRA"
B:      .WORD   ERRB           ;(B) = HEX VALUE, ERROR CODE "ERRB"
           .
           .
           .
```

In this illustration, ERRA and ERRB are the symbolic names of two error codes. Locations A and B contain the numeric values which will be found in AC0 should errors ERRA or ERRB, respectively, happen to occur; in that case, the above statements will cause a branch to either location LOCA or LOCB. Not shown here, they would be the entry points of error handling routines for the two possible cases. Refer to the end of this chapter for another example of the procedure outlined above.

RETURNS

There are three possible outcomes after a system call: no return, normal return only, and both error and normal returns.

- No return (control is never returned to the user task):

```
CALL
.WORD entry
```

- Normal return only (error return not possible, control returned to the next available location):

```
CALL
.WORD entry
normal return location
```

- Both error and normal returns (the next available location is an error return, the one following is a normal return):

```
CALL
.WORD entry
error return location
normal return location
```

An example of system calls containing one call for each type of return is shown on the following page. The call to TEND has no return and is used to terminate the task. SUSP has only a normal return with no input or output parameters, and suspends the task for one scheduler pass. GBF obtains a buffer from the system buffer pool; it requires an input in AC1 and provides an error code in AC0 if the error return is taken, or outputs in AC1 and AC2 if the normal return is taken. Potential error codes from GBF are ERBIS (illegal size requested) and ERBNA (no buffers available).

```
        .TITLE  SAMPLE
        .GLOBL  TEND,SUSP,GBF     ;SYSTEM ENTRY POINTS
        .GLOBL  ERBIS,ERBNA       ;ERROR CODES FROM GBF
        .FORM   CALL,16(X'2C03)   ;DEFINE CALL OPERATOR
          .
          .
          .
        CALL                      ;SUSPEND FOR ONE SCHEDULER PASS
        .WORD   SUSP
          .
          .
          .
        LI      1,1               ;BUFFER SIZE
        CALL                      ;GET A BUFFER BY
        .WORD   GBF               ;CALLING SYSTEM ENTRY GBF
        JMP     ERR               ;ERROR EXIT
        ST      2,BUFLOC          ;NORMAL EXIT
        JMP     END
          .
          .
          .
ERR:    SKNE    0,IBS             ;TEST FOR ILLEGAL SIZE
        JMP     BADSIZ            ;JUMP IF ILLEGAL SIZE
        SKNE    0,NBA             ;TEST FOR NO BUFFERS AVAIL
        JMP     NOBFR             ;JUMP IF NONE AVAILABLE
          .
          .
          .
IBS:    .WORD   ERBIS             ;ERROR CODES FROM GBF
NBA:    .WORD   ERBNA
          .
          .
          .
END:    CALL                      ;TERMINATE THIS TASK
        .WORD   TEND
        .END                      ;END OF ASSEMBLY
```

Chapter 5

MEMORY MANAGEMENT

System II provides dynamic memory management services for application use as well as to support system services. Memory is divided into space for System II, a buffer pool of size selected at system generation, and partition space. Available partition space is determined during system initialization by testing all possible memory. In a J100 with extended memory all possible memory banks are tested.

In a system with extended memory, the lowest physical memory address space, 0 - X'7FFF (called bank zero) is permanently resident; i.e., cannot be switched. The higher address space X'8000 - X'FFFF is switched between banks 1 through 7 as required. System II always resides in lower memory (bank 0).

SYSTEM BUFFER POOL

The system buffer pool is allocated immediately after the system is booted and the size of available memory has been determined. In a standard J100 or J500, it is placed at the end of the first contiguous segment of memory or just below the lowest address CRT refresh area if the memory extends that far. In a J100 with extended memory, the buffer pool resides in bank 0 (address space 0 - X'7FFF) after System II. The size of the buffer pool area is defined at system generation time. It can be redefined without performing a new system generation by using the CHBUFS utility and rebooting the system.

BUFFER SIZES

Buffers of 16, 32, 64, 128, 256, 512, or 1024 (decimal) words are available. The buffer manager breaks down larger buffers as necessary when smaller buffers are requested and recombines them as the smaller buffers are returned.

BUFFER OWNERSHIP

When a buffer is requested by either the system or a user task, the size of the buffer and the job number of the task requesting it are noted in a special table called the BCT (Buffer Control Table). This information is used when the buffer is returned to verify proper ownership and to determine its size. It is also used to release all buffers belonging to a job upon termination.

BUFFER MANAGEMENT CALLS

GET A BUFFER (GBF)

```
CALL
.WORD GBF
error return
normal return
```

Entry Parameters:	AC1 =	Requested buffer size in words, divided by 16; values must lie in the range 1-64.
Exit Parameters:	AC0 =	Error code, if error return is taken.
	AC1 =	Size of buffer actually assigned, divided by 16 (1, 2, 4, 8, 16, 32, or 64).
	AC2 =	Location of buffer.
Error Codes:	ERBIS	Requested buffer size illegal: (AC1) at entry less than 1 or greater than 64.
	ERBNA	Insufficient contiguous space available.

GBF removes a buffer from the pool of available buffers and assigns it to the current job. The size of the requested buffer is specified as an integral number of 16-word segments, so that a buffer of n words is requested by calling GBF with (AC1) = n/16. Permissible values in AC1 are 1 to 64, corresponding to buffers of 16 to 1024 words, respectively.

Due to the nature of the algorithm used to assign and recombine available space, only blocks whose size is an integral power of 2 are actually allocated, that is, blocks of 16, 32, 64, 128, 256, 512, or 1024 words. Consequently, GBF automatically increases a requested size that does not meet this criterion to the next higher even power of 2. The increased size, divided by 16, is then returned in AC1.

The following table gives some examples of this procedure:

Buffer Size Desired (words)	Entry AC1 Value	Number of Words Allocated	Value Returned in AC1
16	1 min	16	1
32	2	32	2
48	3	64	4
64	4	64	4
80	5	128	8
96	6	128	8
112	7	128	8
128	8	128	8
144	9	256	16
.	.	.	.
.	.	.	.
.	.	.	.
256	16	256	16
272	17	512	32
.	.	.	.
.	.	.	.
.	.	.	.
512	32 max	512	32
.	.	1024	64
.	.		
.	.		
1024	64 max		

FREE A BUFFER (FBF)

```
CALL
.WORD FBF
error return
normal return
```

Entry Parameters: AC2 = Buffer address.

Exit Parameters: AC0 = Error code, if error return is taken.

Error Codes: ERBIA Illegal buffer address in AC2.

ERBNJ Buffer is already unassigned or belongs to another job.

FBF returns a buffer to the pool of available buffers. It verifies that AC2 contains a valid buffer address and, if so, that the buffer specified is assigned to the job calling FBF.

PARTITION ADDRESS SPACE

Partition address space consists of those areas of implemented main memory which are not used for the operating system, buffer pool, terminal display buffers and the APL ROM area.

Partition address space is dynamically segmented into partitions by system calls. Requests for partitions are satisfied on a first-fit basis, proceeding from low memory to high unless a high to low search is requested (see GPA below and CHATR utility). Partitions are primarily used to hold programs being executed by the operating system. For this purpose, a partition is allocated at job initiation time and deallocated when no further jobs are executing the program residing in the partition. A partition may consist of both a base sector (BSECT) and a top sector (TSECT) portion. The TSECT portion may be requested in any size; however, all allocations of TSECT space are rounded up to the nearest non-zero integer multiple of 16. Both BSECT and TSECT portions are contiguous within themselves.

The maximum number of partitions that may be allocated at one time is 100. Other uses for partition address space include the allocation of a data area to support a Data-Rite, Report-Rite, or a BASIC language user (in both generative and interactive mode). The System II sort/merge program and other utilities also allocate partitions to provide working space. Partition address space is also required for certain device drivers which are not system resident (for example, line printer and magnetic tape).

As a general rule, system buffers should be used for small segments of memory instead of partitions, since fragmentation of partition address space should be kept to an absolute minimum.

The operating system keeps track of the ownership and usage of partitions by associating them with jobs. An active partition is in use by at least one job and possibly more (for example, a multi-user program). At the termination of a job, the system automatically reduces the use count for all partitions used by the job. Any partition with a zero use count may be recombined into available partition address space when needed.

PARTITION MANAGEMENT CALLS

GET A PARTITION (GPA)

```
CALL
.WORD GPA
error return
normal return
```

Entry Parameters: AC0 = BSECT length or 1's complement of BSECT length.
AC1 = TSECT length.

Exit Parameters: AC0 = Error code, if error return is taken. On normal return, BSECT area origin.

AC1 = TSECT area origin.

AC2 = Partition number.

Error Code: ERNPA No partition available.

GPA allocates a partition consisting of a BSECT area and a TSECT area. The BSECT area will be exactly the size specified; the TSECT area will be rounded up to a multiple of 16 words. Either of the requested BSECT or TSECT lengths may be zero. A minimum of 16 words of TSECT area are always allocated. The partition is allocated from the partition address space of memory which was determined at system boot time. The system looks for an unused slot by scanning from low addresses upward. If (AC0) > 0, the lowest available partition will be assigned. If (AC0) < 0 (1's complement of BSECT length) the highest available partition will be assigned. Partitions containing programs which are not in use are freed as necessary to produce an available slot large enough to accommodate the requested partition. The TSECT area of a partition will always have an address that is an integer multiple of 16. The BSECT area may begin on any address between X'12 and X'FF. The first word of the BSECT and TSECT areas will contain the respective lengths of the partition. In a system with extended memory, GPA will allocate a partition in the same memory bank as the calling program.

All partitions assigned to a job will be reclaimed by the system at job termination if not freed beforehand. A partition may be freed by a call to FPA.

FREE A PARTITION (FPA)

```
CALL
.WORD FPA
error return
normal return
```

Entry Parameters: AC2 = Partition number or partition TSECT address.

Exit Parameters: AC0 = Error code on error return. All registers are restored on a normal return.

Error Code: ERPNJ Partition not assigned to current job.

FPA frees a partition that was previously allocated to a user. If the partition is not assigned to the job making this request, the error return is taken.

(THIS PAGE INTENTIONALLY BLANK)

Chapter 6

JOB INITIATION AND TASK SCHEDULING

JOBS

A job is a collection of one or more tasks which are all part of the same program. The maximum number of concurrent jobs is 100 (decimal). Each job can obtain up to 30 partitions. The terminal from which a job is initiated becomes its message terminal; the association of a message terminal with a job persists until explicitly dissolved by the job.

A job initiated in response to a user command entered at a terminal is a primary or first-level job; it has no predecessor job. One initiated from within a program by means of a LINK call (explained below) is a secondary job; its predecessor is the issuing program.

When a job is created, it consists of a single task, which is started by the Job Initiator. The Job Initiator's function is to cause the program to be loaded into a partition. If the program is not sharable for execution, a new copy must be loaded each time it is called. If the program file is sharable for execution, subsequent commands calling for it will not cause it to be reloaded, so long as an available copy remains in memory. In some circumstances in a system with extended memory multiple copies of sharable programs may be loaded in different banks. See the BANK utility in the System II Utilities manual.

While a job starts as a single task, that task may in turn create others, which become part of the same job. As the tasks execute, they may obtain system resources, which become attached to the job and are shared by all the tasks constituting the job. A job terminates when all its tasks have either terminated or aborted. At that point, the system closes any open files, releases any system allocated buffers still attached to the job, releases any partitions assigned to the job, releases the task control block, and, if necessary, returns the message terminal to control mode.

In addition to the BSECT and TSECT areas in the partition assigned to the program, each task has access to a fixed four-word area in base page memory known as "task page zero space." These four words can be referenced directly by the global system symbols TPZ0, TPZ1, TPZ2, and TPZ3, which

should be declared as external. All tasks use the same four-word area; interference is eliminated by saving the contents of these locations elsewhere when switching among tasks.

The system file SYSSYM.RB contains the definitions for the global symbols TPZ0-TPZ3. It should be included in a link edit of the program.

JOB INITIATION

The initial task of a job receives the following parameters when it begins execution:

AC0-AC1 = Number of words in the TSECT portion of the partition containing the program.

AC2 = Start address.

AC3 = Address of 64-word buffer containing the command.

TPZ0-TPZ3 = Terminal identification in ASCII characters, stored two per word with trailing nulls (X'00).

The first D'41 words of the command-line buffer contain the command line itself; the remaining 23 words may be used by the application program. After processing the command line, the program may use the entire buffer, or release it by calling FBF. While job termination will free the buffer if it has not already been released by the program, a job that runs for very long after processing the command line should release the buffer explicitly as soon as possible.

In most cases, the terminal initiating a job is left attached to the job until the job terminates. However, it is possible for the job to return the terminal to command mode before it terminates. In this event, the terminal can be used to initiate a new job before the first has terminated. A user can continue to initiate new jobs as soon as each releases the terminal. Although a job may detach from it, the terminal continues to function as its message terminal. Messages generated by the job will go to that terminal if it is in command mode and the job was not initiated by a LINK. Having detached from its initiating terminal, a job can open another, which then becomes its message terminal.

When a job is initiated, its task page zero space (TPZ0-TPZ3) is set to the name of the terminal it is to use, referred to here as its TPZ terminal; it need not be a computer terminal. Thus a remote terminal (via the utility MONITR) can invoke those utilities which limit their terminal use to sequential I/O calls.

If the command to initiate the job gives the name of a terminal within brackets, that terminal becomes the TPZ terminal. If it is not a computer terminal, then it is attached to the job.

If the command does not specify a terminal, then the TPZ terminal is determined by the first of the following conditions which is met:

1. The job has been started by a predecessor job having a non-computer terminal attached to itself. In this case, the job's TPZ terminal is that non-computer terminal.

2. The job is a foreground job. In this case, the TPZ terminal is the job message terminal.

3. In all other cases, the job does not have a TPZ terminal, and its task page zero space is set to zero.

TASKS

A task is the focal point for CPU scheduling. The purpose of the system scheduler is to equitably share the CPU among whatever tasks exist at each point in time. Each task represents an independent CPU process. These independent processes generally run asynchronously with regard to one another, but in some cases synchronization between tasks is required. The system provides several mechanisms to support this requirement.

Conceptually, every task acts as if it has its own CPU since the system simulates a virtual CPU for each. This is done by switching control among the active tasks, giving each of the tasks in succession a small slice of the available CPU time. When switching from one task to another, the contents of the CPU registers for the current active task are saved and the previously saved contents of the registers for the next task are loaded back into the CPU registers. Each task also has its own contents for the task base page locations TPZ0, TPZ1, TPZ2 and TPZ3, which are swapped when switching among tasks.

When a job is created, it consists of a single task. That task may in turn use system calls to create other tasks, which become attached to the same job. While each task has its own contents for the CPU registers and task base page locations, other resources can be shared by the various tasks comprising a job. Resources that can be shared are the partition (TSECT and BSECT), buffers and files.

SYSTEM SCHEDULER

The sequencing of task execution is controlled by the system scheduler. Its primary responsibility is the allocation of CPU time to the various tasks. There are three ways that the CPU can be taken away from a task:

- The task voluntarily gives up the CPU via a suspend call or another call which suspends (most I/O calls suspend).

- The task has exhausted its current time slice.

- A higher priority task needs to run.

There are two basic kinds of suspend requests that a task can make:

- Give up the CPU until a specific event occurs which is denoted by a flag posted in a memory location (SUSC, SUSN, SUSZ, SUST calls).

- Give up the CPU for now, and put this task at the bottom of the scheduler queue, but marked as ready to run (SUSP call).

Whenever a task gets suspended, the scheduler puts that task onto one of three priority queues: high, medium and low. The high priority queue is used only for the system character input distributor task (CID). The medium priority queue is for tasks which have suspended by a SUSC, SUSN, SUSZ, or SUST call. The low priority queue is for tasks suspended by a SUSP call or a time slice.

The tasks on the various queues are linked in the order in which the suspends occurred. A task on a queue can be in one of two states: ready to run, or waiting for an event to occur. When the scheduler must select a task to execute, it searches the queues in priority order (high, medium and low). The first task it encounters that is ready to run is the one that is executed. In order for a task on the low priority queue to get executed, there can be no tasks on the high or medium priority queues ready to run. Tasks waiting for an event have a timer which is decremented every clock period. If the timer runs out before the event occurs, the task becomes ready to run, but is instead started up at a time-out location.

A task will not be suspended except by program request when it is running in system code. A task running in user code may also prohibit time slice suspends by making a special scheduler call SUSX (suspends not allowed). This prohibition remains in effect until the next system call is made, which may be the complementary SUSA (suspends allowed).

When a task is activated, an internal timer is set to limit the time the task may run. The limit is two clock periods, which are 0.1 second each. However, since the first clock interrupt can occur anywhere from 0 to 0.1 second after the task is activated, the time slice may in fact be anywhere from 0.1 to 0.2 seconds. If the task has not suspended or terminated before the timer runs out, the system will suspend it. However, if the task has made a system call when the timer runs out, suspension is postponed until control is back in the application program.

SHARABLE PROGRAMS

A sharable program is a reentrantly coded program which has the "sharable for execution" attribute set in the disk catalog entry. The primary characteristic of reentrant code is that there is no local storage of variable information; otherwise, there are no particular restrictions on what a sharable program may do. Like a nonsharable program, the first time a sharable program is called, it is loaded into a partition. Unlike nonsharable programs, however, a subsequent call for it will not reload it from disk if it is still in memory. Rather, a new job calling for it simply begins executing at the start address of the program. It is passed startup

parameters in the manner described earlier. There are some exceptions to this in J100 systems with extended memory. See BANK utility in the System II Utilities manual.

JOB AND TASK TERMINATION

A job can be terminated in any of three different ways:

- If all the tasks constituting a job terminate normally by a TEND (Terminate a Task) system call.
- If any task makes an ABT (Job Abort) system call. This aborts each of the job's tasks that do not have an abort control routine.
- If the job is attached to a terminal and the user at the terminal presses <CANCEL>. This aborts each of the job's tasks that do not have an abort control routine.

The job is not completely terminated until the last task in the job has terminated. Once the job abort flag has been set, the system scheduler is responsible for terminating all the tasks in the job. This can be a complicated procedure since a task cannot be terminated while it is making a system call. Thus, the scheduler must wait until the task returns from any system call and is back in the application program.

When the last task in the job terminates, the system closes any files still open and releases any resources still attached to the job, such as system buffers and partitions. If necessary, the job's message terminal is returned to command mode.

DEADLOCK AVOIDANCE

A deadlock is defined as a condition in which all the tasks in the system are making system requests that cannot be satisfied. When a deadlock occurs, it is often the case that the requests of any single task could be satisfied if that task were running by itself. The problem arises because the system cannot predict which resources each task will request, and must satisfy requests on a first-come first-served basis. The result can be that no one task gets all of its requests satisfied.

Another type of deadlock occurs when a task sets an interlock flag and then inadvertently fails to clear it. Other tasks may then wait indefinitely for the interlock to be cleared. This situation is usually caused by a software bug or design error and should not occur after a program has been thoroughly checked out.

The system makes no effort to prevent deadlocks of either type. Since all resource requests have a "Not Available" error return, it is left to the application program to recognize that a resource deadlock may have occurred and to take appropriate action.

Once a resource deadlock has occurred, the only solution is for some of the tasks to return resources they have already obtained. This can be done either by voluntary action on the part of the program, or by users at terminals aborting some jobs.

In order to avoid deadlocks, application programs should observe the following rule: resources granted by the system should be retained on an all or nothing basis - that is, if any resource request is refused, all other resources should be returned to the system (rather than held until a denied resource becomes available). After an appropriate delay, the program can attempt to acquire the resources again. Those resources hardest to acquire should be obtained first. The following order for requesting resources is suggested:

1. Devices such as printers, paper tape readers, etc.
2. Disk files
3. Buffers

SCHEDULER REQUESTS

The remainder of this section describes various system calls for controlling the execution of tasks. TASK, TEND and ABT create, terminate and abort tasks. The ABTC call specifies a recovery address for jobs which have been aborted. SUSP, SUSC, SUSZ, SUSN and SUST suspend tasks until a specified condition occurs. The SUSX and SUSA calls disable and enable time-slice suspensions.

All of the calls for task suspension, except SUSP, require a timer value to be specified in register AC1. The value is in units of one-tenth of a second. A timer value of -1 (X'FFFF) is treated as infinite. If the timer runs out before the specified event occurs, the task will be restarted at the timeout return address.

Recall that in those calls where an error return is possible, the error code will be found in AC0. Refer to Chapter 4 for details of the possible outcomes of system calls.

SCHEDULE A TASK (TASK)

```
CALL
.WORD TASK
error return
normal return
```

Entry Parameters: AC2 = Entry point address of task.

Exit Parameters: None.

Error Codes: ERBNA No buffer available.

TASK builds a task control block (TCB) for a new task and places it on the scheduler low priority queue. The new task will be attached to the same job as the task making the request. The module whose entry point is supplied in AC2 must be present in memory; no checking is performed on this input. The only error condition is that no memory is available for the TCB.

The initial contents of the CPU registers for the new task will be the same as the contents of the registers for the current task when the TASK call is made. The initial contents of the 4-word task page zero area will be the same as that of the current task.

TERMINATE A TASK (TEND)

```
CALL
.WORD TEND
```

TEND terminates the task making this call. If there are other tasks still active in the same job, the only action taken in this call is to release the task control block. If this is the last task in the job, the system will close any files still open, release any disk sector blocks still assigned to the job, release any buffers still attached to the job and decrement the job count of the partition used by the job. TEND does not return to the user program.

JOB ABORT (ABT)

```
CALL
.WORD ABT
```

The ABT system call sets the job abort flag, which causes all tasks belonging to the job to be aborted. When the last task is terminated, any resources still attached to the job are released. A task that calls ABT will terminate with the abnormal termination message. It will not go to its abort control handler (if any), set by calling ABTC.

ABORT CONTROL (ABTC)

```
CALL
.WORD ABTC
normal return
```

Entry Parameters: AC0 = Abort return address.

Exit Parameters: AC0 = Old abort return address.
AC1-AC3 = Unchanged.

The ABTC call specifies an address to which control is to be transferred when <CANCEL> is pressed. When the abort occurs, the location where the program was interrupted is pushed onto the stack and control is transferred (via a jump) to the abort return address. This abort recovery only applies to the particular task making this call. If this call is made with AC0 = 0,

the task will simply be terminated when the job is aborted. If this call is made with AC0 = -1, aborts on this task are inhibited. Aborts on a task are also inhibited after an abort recovery has occurred. The job abort flag is cleared on an ABTC call.

In a job with several tasks, each task that wants to execute an abort routine must make a separate call to ABTC. When <CANCEL> is pressed, those tasks will each receive control in arbitrary order; the others will be KILLed.

Note that after <CANCEL> is pressed, if any task makes an ABTC call, the abort flag will be effectively cleared for all tasks. This means that any task which has not yet gone to its abort recovery routine will not do so and will go on executing as though <CANCEL> was never pressed.

KILL has the same effect as <CANCEL>. Note that a job can be KILLed even when it is not attached to any terminal.

Causing a STFL or JZER panic task abort has the same effect as <CANCEL>, except that the task causing the STFL or JZER is terminated without going to its abort recovery address.

SUSPEND UNTIL NEXT SCHEDULER PASS (SUSP)

```
CALL
.WORD SUSP
normal return
```

SUSP suspends the current task by placing it at the end of the low priority scheduler queue in a "ready to run" condition. The scheduler will activate the task again on its next pass through the queue.

SUSPEND UNTIL LOCATION CHANGES (SUSC)

```
CALL
.WORD SUSC
timeout return
normal return
```

Entry Parameters: AC0 = Old value in location.
AC1 = Timer value in units of 0.1 second.
AC2 = Location address.

Exit Parameters: None.

SUSC suspends a task until the location given in AC2 contains a value different from the value in AC0, or until the timer runs out.

SUSPEND UNTIL LOCATION ZERO (SUSZ)

```
CALL
.WORD SUSZ
timeout return
normal return
```

Entry Parameters: AC1 = Timer value in units of 0.1 second.
AC2 = Location address.

Exit Parameters: None.

SUSZ suspends a task to wait for the location specified in AC2 to become zero, or the time given in AC1 to elapse. SUSZ provides the ability to wait for some other task to post a communication word with a zero. This type of suspend is used by some operating system routines, but is also available to user programs.

SUSPEND UNTIL LOCATION NON-ZERO (SUSN)

```
CALL
.WORD SUSN
timeout return
normal return
```

Entry Parameters: AC1 = Timer value in units of 0.1 second.
AC2 = Location address.

Exit Parameters: None.

SUSN suspends the task until the location given in AC2 becomes non-zero, or the time given in AC1 elapses. The use of SUSN is similar to SUSZ.

TEST FLAG AND SUSPEND (SUST)

```
CALL
.WORD SUST
timeout return
normal return
```

Entry Parameters: AC0 = Value of flag (must be nonzero). If this value is zero, SUST uses a non-zero value instead.

AC1 = Timer value in units of 0.1 second.

AC2 = Location of flag.

Exit Parameters: None.

SUST is basically a test and set flag routine. If the flag is zero on entry, SUST sets it (to a nonzero value) and returns to the calling task.

If the flag is not zero, SUST suspends the task until it is zero. Then it sets the flag and returns to the user task.

SUST is the proper way to implement a sequencing interlock for a serially-reusable (but not reentrant) subroutine.

SUSPEND NOT ALLOWED (SUSX)

```
CALL
.WORD SUSX
normal return
```

SUSX prevents the system from suspending a task because its time slice has elapsed. Any subsequent system call will clear the SUSX status. A special call, SUSA, is normally used to clear the SUSX status.

SUSPENDS ALLOWED (SUSA)

```
CALL
.WORD SUSA
normal return
```

SUSA reenables time slice suspends after a SUSX.

LINK TO SECONDARY JOB (LINK)

```
CALL
.WORD LINK
error return
normal return
```

Entry Parameters:	AC0 =	Address of command line.
Exit Parameters:	AC0 = 0	Secondary job executed without error; otherwise,
	AC0 ≠ 0	System error code; if return was made to the error-return location, error code indicates that secondary job not started; if return to normal-return location, error code is from secondary job. (See RETURNS, Chapter 4).
Error Codes:	ERCROP	Message terminal open

LINK causes a secondary job to be created and started, using the command line addressed by AC0. The command line must be in standard ASCII format packed two characters per word, beginning with the name of the program to be run, and ending with a line terminator character if less than 80 characters long. The # (Debugger invocation, see page 3-1) is ignored.

The task making this call is suspended until the secondary job has terminated normally, aborted, or executed an RELJ call.

If the secondary job cannot be successfully started, the error return from this call will be taken, with AC0 set to a system error code. If the secondary job is initiated properly, the normal return is taken on completion of the job or when an RELJ call is made. When the normal return is taken, AC0 will be zero unless the secondary job called SMSG, FMSG, or ABT. In this case AC0 will contain the system error code.

If the current job is a foreground job and has its message terminal open, LINK returns error code ERCROP. This terminal must be closed because the secondary job is started as a foreground job on this terminal.

RELEASE FROM PREDECESSOR JOB (RELJ)

```
CALL
.WORD RELJ
error return
normal return
```

Entry Parameters: None.

Exit Parameters: AC0 = Error code on error return.
AC1-AC3 = Unchanged.

Error Code: ERCROP Terminal already open.

This call is used to detach a secondary job from the predecessor job which initiated it. The predecessor job will be taken out of the suspended state and allowed to continue running. If the predecessor job was attached to its message terminal at the time of its most recent call to LINK, and either the job calling RELJ has the terminal or no job has the terminal, then the predecessor job is reattached to the terminal. Hence, it must be closed before making this call.

If the job calling RELJ was initiated directly from a terminal, the only effect of the call is to put the terminal back into command mode.

The end result of this call is always that the job is left running in background mode, that is, not attached to any terminal.

When a task with a command file attached to its job calls RELJ, the command file is detached from the job, and the next line of the command file (if any) is used to initiate a job with the command file attached to this new job.

When a task in a primary, foreground job without a command file calls RELJ, the job is detached from its message terminal, and this terminal is returned to command mode. (In the preceding sentence "primary, foreground job" describes the job at the instant when one of its tasks calls RELJ.)

When a task in a primary, foreground job with a command file calls RELJ, the job is detached from its message terminal, and the next line (if any) of the command file is used to initiate an initially primary, foreground job on this terminal with the command file attached to this new job. The terminal is not available for keyboard input until the command file is exhausted.

When a task in a secondary job without a command file calls RELJ, its immediate predecessor job is marked as ready to continue execution, and it is detached from the job whose task called RELJ.

When a task in a secondary job with a command file attached to its job calls RELJ, the next line (if any) of the command file is used to initiate a job with the command file attached to this new job. This new job has the same immediate predecessor job as the job that called RELJ. This predecessor job is not marked ready to continue execution until the command file is exhausted.

Chapter 7

SEQUENTIAL FILE MANAGEMENT

FILE CONCEPTS

System II sequential file management provides file oriented input and output support for all system devices. A file is a collection of related information which is associated with a particular (source or destination) device and identified by a file name.

A sequential file is a continuous data string which is transferred to or from the application program in segments as requested by sequential file calls. The segments are either lines terminated by special line delimiters or strings of a length dictated by the user program. New segments are always added to the end of the file. Particular segments can be retrieved only by reading through the file from the beginning. A sequential file may be associated with any system device; physical differences between devices are handled within the file management routines. Only one file at a time can be assigned to a sequential device such as a printer or a terminal.

A directory-structured device such as a disk may contain many sequential files at once since each platter or cartridge contains a file name directory entry for every file on the particular medium. The directory entry includes the logical file name, the current size and location of the file, the date the file was written, and the file attributes (see CHTR call). A disk file must reside on one volume.

FILE NAMES

A file name is specified to the system when an "open" request is made. This implies that the application program intends to transfer data to or from that file. In the case of sequential devices, the standard device name or device identifier serves as the file name. For disk files, both the device identifier and logical file name must be included unless the disk is the nominal or primary disk. (Refer to Chapter 2, File Naming Conventions.) For example, FP01:ABC refers to the file ABC on the flexible disk unit one. LPT1 refers to a line printer file.

The file names to be used in open requests may be assembled in a user program or may be acquired by the program through keyboard input. The file names may be passed to the program in the command line which invoked the program or may be keyed in later in response to read line requests to the terminal.

A file name must be formatted in a "file name block" to be passed to system routines (required by OPEN, OPNW, OPNR, CREA, DELT, CHTR, and RENAM). (See File Name Blocks in this chapter.) The Command Line Parser (see Appendix B) can be called to format file names for this purpose.

DEVICE NAMES

Each device has a standard name, linking a user's reference to a device with the appropriate system tables and routines to operate it. A list of these names appears in Chapter 2.

FILE CHARACTERISTICS

The characteristics of a file that is a device are defined during system generation. The user may exercise some runtime control over the characteristics of a disk file by changing the file's attributes in its directory entry.

The characteristics defined for the system devices inform the file management routine of special processing requirements. These characteristics are described in System Generation manuals.

FILE ATTRIBUTES

The attributes of a file define the manner in which it can be used. Non-disk devices have predefined attributes; the attributes for disk files can be defined and changed by users with the CHTR call or the CHATR utility. The CHTR call and the CHATR utility set attributes in the directory entry for the file. Special execution attributes related to execution in a J100 with extended memory can be set by the BANK utility. These attributes are set in the title block of a binary module. Directory attributes and their meanings are as follows:

- Attribute Protected

  The current attributes cannot be altered.

- Execution Sharable

  The program in this file may be shared by multiple users. This implies a reentrant program.

- Read Sharable

  The file may be read by multiple users simultaneously.

- Write Sharable

  The file may be written into by multiple users simultaneously.

- Read Only

  The file may be read but not written into.

- Write Only

  The file may be written into but not read (appears as an attribute for output devices such as the printer).

- Permanent

  The file cannot be deleted or renamed. Permanent does not imply read only; you can write to a permanent file.

- Top Loading

  The file, a binary program, should be loaded in the highest available partition, rather than the lowest; recommended for jobs which will be used continuously (in order to minimize the fragmentation of partition space).

- Secure File

  Access to the file is restricted.

System utilities, but not user programs, can set the following attributes:

- Hash File

  The file is keyed.

- Duplicate Keys

  The same key may appear in more than one record.

- Batch

  The file is a batch file (a special file structure used by Data-Rite).

- Permits on File

  The file is secure, but the owner has permitted someone else access to it.

For non-disk devices, the predefined attributes always include Attribute Protected and, as appropriate, either Read Only or Write Only; no sharing is set for devices such as terminals and printers.

SHARING FILES

Files associated with some non-disk devices cannot be shared. The predefined attributes of such a file cause the rejection of requests to share. Since it is possible for the user to define the attributes of disk files, such files may be shared or not according to the current setting of the attributes in the file's directory entry.

In addition to the attributes given in the directory for a disk file, its current availability for sharing is affected by the open requests made by programs using it. The effect of the open on the file's availability for sharing is temporary, lasting only until the file is no longer open. The sharing attributes of a file can only be limited by the open. To expand the sharing, a change attributes command (CHATR) or call (CHTR) must be made. For example, a file defined in the directory as both read sharable and write sharable can be opened with a request for only shared reading. As long as that file is in use (until the close request), it will only be available to additional requests for shared reading, not exclusive reading.

The Execution Sharable attribute has meaning only for binary program files suitable for system loading and execution. When this attribute is set, the system considers the program reentrant and will allow more than one job to use the same copy simultaneously. If this attribute is not set, a second request for the same program will cause a second copy of the program to be loaded.

SYSTEM I/O CONCEPTS

CREATING A NEW FILE

Prior to performing any file input or output, the user must acquire the file through a successful open request. An open will be refused if the file is not known to the system or is not available for the use requested.

A file is known to the system if it is called by a standard sequential device name or if it currently exists in the directory of the disk unit referenced in the file name passed in the open call. To introduce a new sequential file into a directory, a CREATE command must be keyed in from a terminal or a CREA call given in a program. Once the file name entry is created, it will remain in the directory until deleted with a DELETE command or a DELT system call. If the permanent file option is set, the file name entry cannot be deleted.

The availability of a file for a particular use depends upon its attributes and its current use. For sequential devices, the attributes are fixed such that most of these devices are not shared and are available for either reading or writing (or both as appropriate). A request to open a non-sharable device type file that is already open will be refused, as will a request for incorrect use such as opening a printer for reading.

When a sequential disk file is created, no attributes are set. If the file is to be shared or have any other attributes, these must be set with a CHATR command from the terminal or a CHTR call from a program.

OPENING A FILE

The file open calls (OPNR, OPNW, and OPEN) associate a file with the user's job, and the system device on which the file will be accessed. The open call also informs the system of the manner in which the file will be accessed and whether the user wishes to share the file. The open will be refused if the file does not exist or the intended use of the file cannot be allowed because of the file's attributes or a previous open.

The share attributes of a disk file are combined with the share flags declared at the first open for that file to produce a file "share status". As long as any opens are in effect, the system maintains this dynamic share status which is a composite of share flags for all opens for the file and the original file attributes. When a file has been closed by all its users, the dynamic share status is dropped by the system.

The initial open for a file is the most important as it causes the system to create and maintain the share status. The initial share status can be determined from the table shown below. It shows the resulting share status given the file attributes and the share flags declared at the initial open.

FILE SHARE STATUS TABLE

File Attributes \ Initial Open Share Flags	NS	RS	WS	BS
NS	NS	NS	NS	NS
RS	NS	RS	NS	RS
WS	NS	NS	WS	WS
BS	NS	RS	WS	BS

NS = no share
RS = read share
WS = write share
BS = read/write share

Open requests subsequent to the first must declare share flags that match the share status determined from the above table or the open will be refused.

When the file is opened successfully, the address of a User File Table (UFT) is returned to the calling program. This UFT address must be passed to the data transfer routines and the close routine. The user is responsible for retaining the UFT address and passing it with each of these calls.

DATA TRANSFER

Two major levels of sequential data transfer support are provided by System II. This support allows data to be transferred as a block of specified size or as a block of unspecified size ended by a line terminator. The system provides devices and disk files with identical support, which allows device independent I/O by application programs. All characters are considered to be 8-bit bytes.

All data transfer calls provided by the system require a User File Table address to be specified. A UFT is built by system routines during open processing. Therefore, in order to transfer data, an open must have been successfully completed for that file.

Transfer of character blocks of specified length is termed "read/write sequential." In this mode, a specified number of characters (to a maximum of 32,767) is transferred between the file and the user area. The character string must be packed two characters per word (left to right) for a write. Similarly, for a read, the characters will appear in the user work area in this form. On a read of an odd number of characters, the low order character of the user area will be null.

Transfer of an ASCII line of unspecified length is termed "read/write line". In this mode, characters are transferred until a line terminator is encountered in the character stream. On a read this is subject to a maximum count. TAB characters (X'09, ASCII HT) are expanded to an appropriate number of blanks during input. Compression on output is left to the user program.

A line terminator is defined to be any 8-bit character less than or equal to X'0F except X'09 (TAB).

Transferred character strings are packed two characters per word, left to right, in the buffer area. When a line with an odd number of characters is read, the low order character of the user area will be lost just as if an even number had been transferred (the extra is a null character).

Once a file has been opened, data transfer may proceed with either read/write line or read/write sequential system calls. These calls may be arbitrarily mixed since they are completely compatible. An application program should exercise caution when reading a file in "line" mode if it were written using the "sequential" mode, to ensure that terminators were properly included at write time. Each character is treated as an 8-bit byte which is read or written without change unless translation has been requested or TABs are expanded.

For read line calls, where data transfer proceeds subject to a maximum count, if the count is exhausted before a terminator is encountered, an error return will be made with the partial line in the user's work area. A subsequent read request will continue with the balance of the line. No data will be dropped, but tab expansion may be incorrect (see RDL call).

End-of-file is indicated with an error return. When such a return occurs, some data may have been transferred, so the return count should always be inspected. For devices, end-of-file will occur when the device is timed out for no response. For disk files, end-of-file is implied after the last character written to the file.

The system data transfer routines support shared files on disk. File attributes have to be properly defined in order to successfully open files in share mode. A file being shared for reading allows multiple, independent reading to occur on the file. Each read process will access the file as if it were the only read occurring on that file.

A file being shared for writing will similarly appear to each write process as if it were not being shared, except the contents of the file will be an interleaving of the data output by the independent write processes. In such a file, all data provided in a single system write call will always appear together.

The system also allows sequential disk files to be shared for both reading and writing which permits multiple I/O processes to be concurrently reading and writing the same file. In such a situation, the write processes behave as if there were no read processes operating on the file. The read processes are only indirectly aware of the write processes. A read request can access all data provided to the file up to the instant of the read call. Data being added to the file via an in-progress write process is not available until the write process is completed. In order to intersperse both reading from, and writing to, the same file, the file must be opened twice; once for reading and again for writing, thus maintaining two UFTs.

In a multi-task program environment, the sharing of a UFT between tasks is generally not permitted. Specifically, a task may not call the system with a UFT that is in use by another task in a system call that is yet to be completed. Caution must be exercised since failure to adhere to this rule may lead to system failure.

CLOSING A FILE

When all data transfers are complete, the CLOS call is used to disassociate the file from the job, perform any final device specific functions (such as page ejection on a printer), free the system resources used for processing the file and, in the case of disk files open for writing, update the directory to reflect the current size of the file and the current system date.

DISK DIRECTORY FUNCTIONS

The system calls Create (CREA), Delete (DELT), Rename (RNAM), and Change Attributes (CHTR) operate on disk file directory entries. These calls are all rejected if the file named is currently in use (open). CREA places an entry in the directory for the file name passed to it. DELT removes the file name given and releases the disk space used by the file. RNAM changes the name of an existing file. CHTR alters the file attributes in the directory entry for the referenced file.

FILE NAME BLOCKS

The internal representation of file names differs from the external form used in commands. The Command Line Parser (Appendix B) converts external form to internal form. The following conventions apply to file name blocks:

Each part of a name (prefix, root, extension) must be left justified with a filler of trailing nulls (X'00).

The contents of the root and extension words determine the meaning of the prefix words. If the root and extension words specify a device, then with a single exception noted below, the prefix is irrelevant. If the root and extension words do not specify a device, then the prefix identifies a disk, as follows:

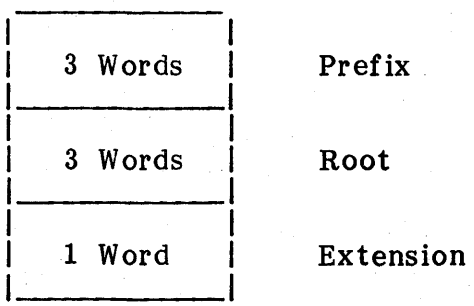
- A prefix consisting of a dollar sign, left justified with trailing nulls, specifies the primary disk;
- Three words of hexadecimal zero specifies the nominal disk;
- Otherwise, the prefix explicitly names the disk.

The single exception referred to occurs when the root and extension is the name of one of certain communications devices. In this case, the prefix words may contain user identification characters, which the communications device appends to each message to identify its source.

If the root and extension words contain the name of a computer terminal, then the prefix is irrelevant. This fact allows the task page zero space (TPZ0-TPZ3), as set by job initiation, to be used in place as the last four words of a file name block specifying the job's TPZ terminal, thereby obviating the need for a file name block.

If the root and extension words contain the name of a nondisk device other than those covered above, then the prefix is not used and not examined. In the future, however, a new device may assign a special meaning to the prefix, just as certain communications devices have done. Consequently, the three-word prefix segment should be zeroed when it is not used.

File Name Block



FILE I/O CALLS

CREATE A FILE (CREA)

```
CALL
.WORD CREA
error return
normal return
```

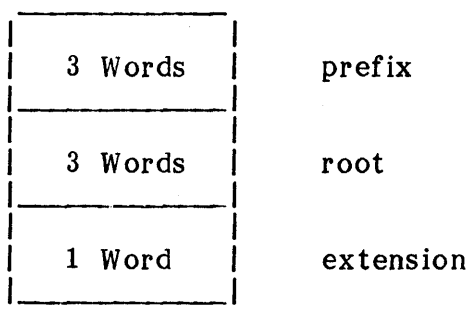
Entry Parameters: AC0 = Pointer to file name block.

Exit Parameters: AC0 = Error code on error return.
AC0 = As input if normal return.

Error Codes:

ERBNA	No buffer available.
ERFNA	File (in this case, the disk drive) not available now.
ERFNE	File name already exists.
ERFNI	Illegal file name.

File Name Block



See File Name Blocks, page 7-8, for formats of file name blocks.

CREA makes a new directory entry and this must be done before any other access to the file is allowed. CREA initializes the new entry to contain the file name and extension, the date, and the first sector in the file. All other fields are set to zero.

CREA checks to make sure that the file name is legal. A CREA call for devices such as printers is permitted, but no catalog entry is built.

OPEN SEQUENTIAL FILE FOR READING (OPNR)

```
CALL
.WORD OPNR
error return
normal return
```

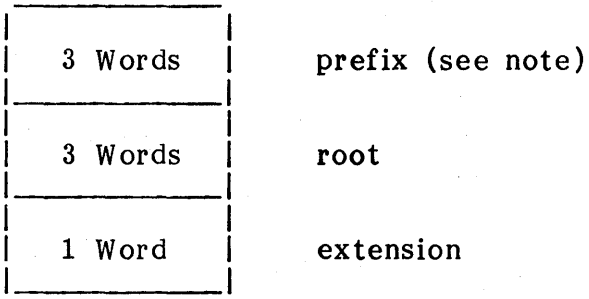
Entry Parameters: AC0 = Pointer to file name block.
AC1 = Share flags.

Exit Parameters: AC0 = Error code on error return.
AC0 = File name pointer on normal return.
AC1 = Share flags.
AC2 = UFT address.
AC3 = Unchanged.

Error Codes:

ERBNA	Buffer not available.
ERDVNA	Device not available.
ERDVR	Cannot load device driver.
ERFAP	File attributes prohibit request.
ERFDSK	File name is a disk.
ERFNA	File not available now.
ERFND	File name device prefix is not a disk.
ERFNF	File not found in directory.
ERFOP	File open options prohibit request.
ERFSH	File sharing request conflicts with use.
ERVLG	User not logged on.
ERVSP	Security prohibits.

File Name Block



See File Name Blocks, page 7-8, for formats of file name blocks.

Note: When opening standard system devices (see Device Identifiers, Chapter 2), the device name should appear in the root and extension of the file name block illustrated above. The prefix field should contain zero.

OPNR opens the specified device, sequential disk file, or hash file for reading, creates a user file table (UFT) and increments the use count in the System File Table. The user passes a request for sharing or no sharing in AC1.

The share flag settings are:

0	no sharing
2	share for write
4	share for read
6	share for both read and write

If the file is not already open, the open routine checks the open request against the file attributes from the directory entry. The request is

refused if the attributes do not allow the file to be read or shared in the manner requested. For details, see the section titled Opening a File on page 7-5.

If the file is already open, this request must conform to the sharing permitted by the previous open.

If the previous open did not allow sharing for read, the request is rejected. In addition, the current sharing request (AC1) must agree with the sharing allowed by the previous open. It is not possible to override the first open's sharing request.

OPEN SEQUENTIAL FILE FOR WRITING (OPNW)

```
    CALL
    .WORD OPNW
    error return
    normal return
```

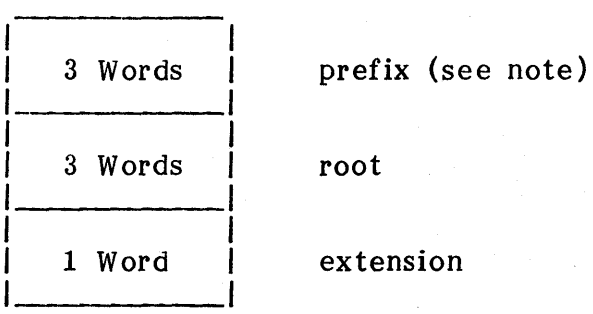
Entry Parameters:
AC0 = Pointer to file name block.
AC1 = Share flags.

Exit Parameters:
AC0 = Error code on error return.
AC0 = File name pointer if normal return.
AC1 = Share flags.
AC2 = UFT address.
AC3 = Unchanged.

Error Codes:

ERBNA	Buffer not available.
ERDVNA	Device not available.
ERDVR	Cannot load device driver.
ERFAP	File attributes prohibit request.
ERFDSK	File name is a disk.
ERFNA	File not available now.
ERFNF	File not found in directory.
ERFOP	File open options prohibit request.
ERFSH	File sharing request conflicts with use.
ERVLG	User not logged on.
ERVSP	Security prohibits.

File Name Block



See File Name Blocks, page 7-8, for formats of file name blocks.

Note: When opening standard system devices (see Device Identifiers, Chapter 2), the device name should appear in the root and extension of the file name block illustrated above. The prefix field should contain zero.

OPNW opens the specified device, sequential disk file, or hash file for writing. The file must already exist before it can be opened for writing (see CREA). If data already exists on the file, any new data is appended to the end. The user passes a request for sharing or no sharing in AC1.

The share flag settings are:

0	no sharing
2	share for write
4	share for read
6	share for both read and write

If the file is not already open, the open routine checks the open request against the file attributes from the directory entry. The request is refused if the attributes do not allow the file to be written or shared in the manner requested.

If the file is already open, this request must conform to the sharing permitted by the previous open.

If the previous open did not allow sharing for write, the request is rejected. In addition, the current sharing request (AC1) must agree with the sharing allowed by the previous open. It is not possible to override the first open's sharing request.

OPEN SEQUENTIAL FILE FOR READING AND WRITING (OPEN)

```
CALL
.WORD OPEN
error return
normal return
```

Entry Parameters:
AC0 = Pointer to file name block.
AC1 = Share flags.

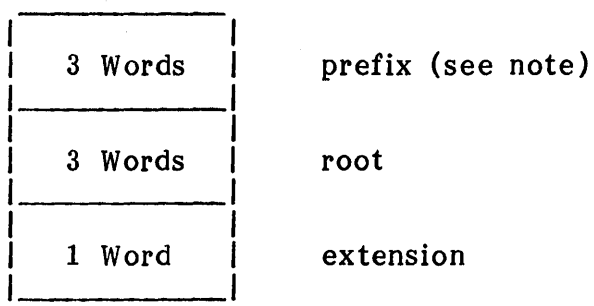
Exit Parameters:
AC0 = Error code on error return.
AC0 = File name pointer on normal return.
AC1 = Share flags.
AC2 = UFT address on normal return.
AC3 = Unchanged.

Error Codes:

ERBNA	Buffer not available.
ERDVNA	Device not available.
ERDVR	Cannot load device driver.
ERFAP	File attributes prohibit request.
ERFDSK	File name is a disk.
ERFNA	File not available now.
ERFND	File name device prefix is not a disk.
ERFNF	File not found in directory.

ERFOP	File open options prohibit request.
ERVLG	User not logged on.
ERVSP	Security prohibits.

File Name Block



See File Name Blocks, page 7-8, for formats of file name blocks.

Note: When opening standard system devices (see Device Identifiers, Chapter 2), the device name should appear in the root and extension entry of the file name block illustrated above. The prefix field should contain zero.

OPEN opens the specified device, sequential disk file, or hash file for both reading and writing. This open can be used for terminals, sequential files, hash files (see Chapter 8), magnetic tape (see Chapter 11), and communication lines. The file must be available for both reading and writing; it cannot be either a read only file or a write only file. If it is already open, both read and write sharing must be allowed.

The share flag settings are:

0	no sharing
2	share for write
4	share for read
6	share for both read and write

If the file is not already open, the open routine checks the open request against the file attributes from the directory entry. The request is refused if the attributes do not allow the file to be read, written or shared in the manner requested.

If the file is already open, this request must conform to the sharing permitted by the previous open.

If the previous open did not allow sharing for read or write, the request is rejected. In addition, the current sharing request (AC1) must agree with the sharing allowed by the previous open. It is not possible to override the first open's sharing request.

READ SEQUENTIAL (RDS)

```
CALL
.WORD RDS
error return
normal return
```

Entry Parameters: AC0 = Work area address.
AC1 = Byte count.
AC2 = UFT address.

Exit Parameters: AC0 = Error code if error.
AC1 = Byte count.
AC2, AC3 = Saved.

Error Codes:

ERBUFT	Bad UFT address.
EREOF	End of file.
ERFOP	File open options prohibit request.
ERICL	Illegal system call.
ERISZ	Illegal size request.
ERSQHF	Sequential I/O call on hash file.
ERSQNS	Sequential I/O not supported on this device.

RDS reads the next block of characters from the file associated with the specified UFT. The data will be packed in the work area, and the number of characters in the block is determined by the byte count specified with the call. This count must be positive and non-zero or an error return will result. If an odd number of bytes are transferred, the right byte of the last word will be set to zero. Translation occurs if requested (see Chapters 10 and 11).

Both the UFT in AC2 and the contents of AC3 which may be arbitrary, will be returned intact after the call. If the normal return occurs, AC0 will contain its original value as well. A byte count will always be returned in AC1 to indicate the actual characters moved. For the error return, AC0 will always contain a system error code.

An end-of-file will cause an error return and will occur if the entire block size requested could not be satisfied from the file. Since an EOF will accompany a short block, the return byte count should be inspected to determine if any data was transferred. Subsequent read calls will also produce EOF returns.

WRITE SEQUENTIAL (WRS)

```
CALL
.WORD WRS
error return
normal return
```

Entry Parameters: AC0 = Work area address.
AC1 = Byte count.
AC2 = UFT address.

Exit Parameters: AC0 = Error code if error return.
AC1 = Byte count.
AC2, AC3 = Saved.

Error Codes:

ERBUFT	Bad UFT address.
ERDSNA	No disk space available.
ERFOP	File open options prohibit request.
ERICL	Illegal system call.
ERISZ	Illegal block size.
ERSQHF	Sequential I/O call on hash file.
ERSQNS	Sequential I/O not supported on this device.

WRS will transfer the specified block of characters from the user work area into the file associated with the specified UFT. The block size must be positive and non-zero for non-BSC devices. For BSC devices a negative count means "perform a control function." The data will be added to the end of any other data in the file and the EOF for any read process on the file will be correspondingly extended. Translation occurs if requested (see Chapters 10 and 11).

Both the UFT in AC2 and the contents of AC3, which may be arbitrary, will be returned intact after the call. If the normal return occurs, AC0 will contain its original value. A byte count will always be returned in AC1 to indicate the actual characters moved. For the error return, AC0 will always contain a system error code.

Substantial time can be saved by writing blocks with an even number of characters. This is because character boundary realignment is avoided.

READ LINE (RDL)

```
CALL
.WORD RDL
error return
normal return
```

Entry Parameters: AC0 = Work area address.
AC1 = Maximum byte count.
AC2 = UFT address.

Exit Parameters: AC0 = Error code if error return.
AC1 = Byte count.
AC2, AC3 = Saved.

Error Codes:

ERBUFT	Bad UFT address.
EREOF	End-of-file.
ERFOP	File open options prohibit request.
ERICL	Illegal system call.
ERISZ	Illegal maximum count.
ERLTL	Line exceeds maximum count.
ERSQHF	Sequential I/O call on hash file.
ERSQNS	Sequential I/O not supported on this device.

RDL reads the next ASCII line from a sequential file and packs it to the user's work area. The copy is terminated by a line terminator character, or by exhausting the maximum byte count, which must be positive and non-zero. If the maximum byte count is exhausted before a terminator is detected, the next RDL will continue with the balance of the long line. Tabs (ASCII HT) are converted to blanks; translation occurs if requested (see Chapter 15).

Both the UFT address in AC2 and the contents of AC3, which may be arbitrary, will be returned intact after the call. If the normal return occurs, AC0 will contain its original value. A byte count will always be returned in AC1 to indicate the actual number of characters moved. For the error return, AC0 will always contain a system error code.

An end-of-file causes an error return; it occurs if the file data is exhausted before either a terminator is encountered or the maximum count is exceeded. The return byte count should be inspected to determine how much data has been moved. Subsequent read calls will also produce EOF returns.

An error code of ERLTL is returned if the user's work area is too short to contain an entire line; in this case, there will not be a line terminator in the work area. Subsequent reads will expand tabs correctly if and only if the maximum byte count for each call is an integer multiple of 8. Satisfying this requirement allows RDL to fully expand each tab when it is encountered and before any error return.

READ LINE QUICKLY (RDLQ)

```
CALL
.WORD RDLQ
error return
normal return
```

Entry Parameters:
AC0 = Address of line buffer.
AC1 = Maximum number of bytes allowed.
AC2 = Address of UFT.

Error Exit:
AC0 = System error code.
AC1 = Number of bytes moved into line buffer.
AC2, AC3 = Unchanged.

Normal Exit:
AC1 = Number of bytes moved into line buffer including the line terminator.

AC0, AC2, AC3 = Unchanged.

Error Codes:

ERBUFT	Bad UFT address.
EREOF	End-of-file.
ERFOP	File open options prohibit request.
ERISZ	Illegal maximum count.
ERLTL	Line too long.
ERSQHF	Sequential I/O call on hash file.
ERSQNS	Sequential I/O not supported on this device.

RDLQ reads the next line from the specified file and sets appropriate values into the specified line buffer. The values set into the specified line buffer depend on the line contents and the device associated with the specified User File Table (UFT).

If the specified maximum byte count is not positive and non-zero, RDLQ returns error code ERISZ. If RDLQ returns error code ERLTL (line too long), the next sequential read call will continue reading the long line. When RDLQ returns error code ERLTL, there is no line terminator in the line buffer.

When RDLQ returns error code EREOF, the number of bytes returned should be inspected to determine if a partial line was returned. Subsequent sequential read calls will also return error code EREOF.

If the device associated with the specified UFT does not support sequential I/O, RDLQ returns error code ERSQNS.

RDLQ From a Sequential File on a Disk or Remote-Disk

RDLQ performs more quickly than RDL because RDLQ does not expand tabs, nor does it compress blanks into tabs. RDLQ does not change the data read in from the file.

RDLQ From a Computer Terminal

RDLQ from a Computer Terminal (CT) is precisely the same as RDL from the CT. The screen contents are moved unchanged from the screen to the specified line buffer. No expansion or translation is done when the data is moved from the screen. However, if the CT has a keyboard translation table, then each key code entered by depressing a keyboard key is translated before it is put on the screen.

RDLQ From a Non-CT Device

RDLQ from a non-CT device is exactly the same as RDL from the device.

Translation occurs if requested. See Chapters 10 and 11.

WRITE LINE (WRL)

```
CALL
.WORD WRL
error return
normal return
```

Entry Parameters: AC0 = Work area address.
AC2 = UFT address.

Exit Parameters: AC0 = Error code if error return.
AC1 = Byte count.
AC2, AC3 = Saved.

Error Codes:

ERBUFT	Bad UFT address.
ERDSNA	No disk space available.
ERFOP	File open options prohibit request.
ERICL	Illegal system call.
ERSQHF	Sequential I/O call on hash file.
ERSQNS	Sequential I/O not supported on this device.

WRL writes the next ASCII line into a sequential file. Data is transferred until a line terminator is encountered in the data stream. The new line will be added to the end of any other data in the file, and the EOF for any read process on the file will be correspondingly extended. Conversion of blanks to TABs is left to the user. Translation occurs if requested (see Chapters 10 and 11).

Both the UFT in AC2 and the contents of AC3, which may be arbitrary, will be returned intact after the call. If the normal return occurs, AC0 will contain its original value. A byte count will always be returned in AC1 to indicate the actual characters moved. For the error return, AC0 will always contain a system error code.

Substantial time savings can be obtained by always writing lines with an even number of characters. This is because character boundary realignment is avoided.

WRITE LINE COMPRESSED (WRLC)

```
CALL
.WORD WRLC
error return
normal return
```

Entry Parameters: AC0 = Address of line buffer.
AC2 = Address of UFT.

Error Exit: AC0 = System error code.
AC1 = Number of bytes moved.
AC2, AC3 = Unchanged.
Contents of line buffer unchanged.

Normal Exit: AC1 = Number of bytes appended to file.
AC0, AC2, AC3 = Unchanged.
Contents of line buffer unchanged.

Error Codes:

ERBUFT	Bad UFT address.
ERDSNA	No disk space available.
ERFOP	File open options prohibit request.
ERSQHF	Sequential I/O call on hash file.
ERSQNS	Sequential I/O not supported on this device.

WRLC appends the specified line to the specified file. The contents of the line appended to the file depends on the specified line contents and the device associated with the specified User File Table (UFT).

WRLC never changes the contents of the specified line buffer.

The line is terminated by any byte less than X'10 except X'09 (tab).

If the device associated with the specified UFT does not support sequential file I/O, WRLC returns error code ERSQNS.

WRLC to a Sequential File on a Disk or Remote-Disk

WRLC compresses the specified line by dropping blanks and inserting tabs at the standard system tab stops (offsets 0, 8, 16, etc. into the line buffer) as appropriate. Then WRLC appends the compressed line to the file associated with the specified User File Table (UFT). The number of bytes appended to the file (including the line terminator) is returned.

WRLC compresses the line as it moves it from the specified line buffer into the sector buffer used for disk I/O. Hence, WRLC does not change the contents of the line buffer.

WRLC to a Computer

WRLC to a Computer Terminal (CT) is exactly the same as WRL to the CT. The keyboard translation table is not used by WRLC. WRLC moves the exact contents of the specified line to the screen of the specified CT. WRLC does not compress or translate the line.

WRLC to a Non-CT Device

WRLC to a non-CT device is exactly the same as WRL to the device.

If the device has a translation table, each byte is translated before it is written to the device. See Chapters 10 and 11.

CLOSE FILE (CLOS)

```
    CALL
    .WORD CLOS
    error return
    normal return
```

Entry Parameters: AC2 = UFT address.

Exit Parameters: AC0 = Error code on error return.
AC1-AC3 = Unchanged.

Error Codes:

ERBNA	Buffer not available.
ERFNF	File name not in directory.
ERFNOP	File not open.
ERSNR	Disk sector not released.

CLOS decrements the use count in the system file table (SFT), deletes the user file table (UFT) for the file, and updates the directory if the file resides on disk and was open for writing.

DELETE A FILE (DELT)

```
CALL
.WORD DELT
error return
normal return
```

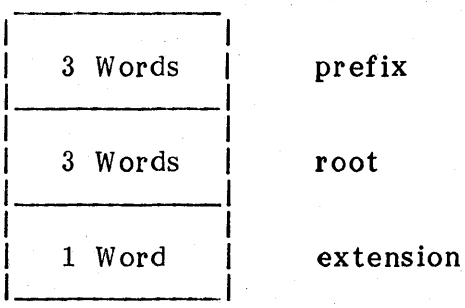
Entry Parameters: AC0 = Pointer to file name block.

Exit Parameters: AC0 = Error code if error return.
AC1-AC3 = Unchanged.

Error Codes:

ERBNA	Buffer not available.
ERFAP	File attributes prohibit request.
ERFNA	File or device not available now (the directory is not available).
ERFNF	File name not found in directory.
ERVOW	Only the owner may.
ERVPP	Permits outstanding.

File Name Block



See File Name Blocks, page 7-8, for formats of file name blocks. DELT deletes a sequential file name from the file name directory and releases any disk space associated with that file. The file cannot be deleted while it is open, and a permanent file cannot be deleted. A hash file cannot be deleted with this call.

RENAME A FILE (RNAM)

```
CALL
.WORD RNAM
error return
normal return
```

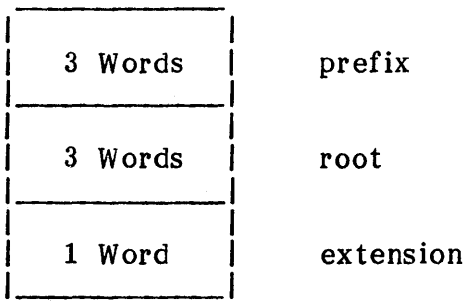
Entry Parameters: AC0 = Pointer to old file name block.
AC1 = Pointer to new file name block.

Exit Parameters: AC0 = Error code if error return.
AC1-AC3 = Unchanged.

Error Codes:

ERBNA	Buffer not available.
ERFAP	File attributes prohibit request.
ERFNA	File or device not available now (the directory is not available).
ERFNE	New file name already exists.
ERFNF	Old file name not found in directory.
ERVOW	Only the owner may.
ERVPP	Permits outstanding.

File Name Block



See File Name Blocks, page 7-8, for formats of file name blocks.

RNAM changes the file name to the new name. The file name cannot be changed while the file is in use. If a file already exists with the new name, the error return is taken. The prefix field in the "new file name block" is ignored.

RNAM checks the attributes in the old name's entry to be sure a name change is allowed. A file that is permanent cannot have its name changed. Since the directory is constructed by applying a hashing algorithm to the file name, after a rename the file will have moved in a catalog list of the disk directory.

CHANGE ATTRIBUTES (CHTR)

```
CALL
.WORD CHTR
error return
normal return
```

Entry Parameters: AC0 = File name block address.
AC1 = Attribute word.

Exit Parameters: AC0 = Error code if error return; else unchanged.
AC1-AC3 = Unchanged.

Error Codes:

ERBNA	Buffer not available.
ERFAP	File attributes prohibit request.
ERFNA	Directory not available.
ERVLG	User not recognized by File Security system.
ERVOW	Only the file owner may call for the action requested.
ERVPP	Permits outstanding.

CHTR is used to change a file's attributes. The file must not be in current use. The word specified in AC1 is used to modify certain bits in the disk directory entry for the file; the prohibited bits (see below) must be zeros. Error returns leave the file's attributes unchanged.

A hash file is always Read Sharable and Write Sharable, regardless of CHTR calls. Similarly, the Secure File attribute, once set, is not affected by a CHTR call unless the Unsecure File bit is 1 in the call's Attribute Word.

It is illegal to make a CHTR call if a file is Attribute Protected, or to set Execution Sharable or Top Loading for a hash file. It is also illegal to process a secure file if the current file security user is not the file's owner, or to unsecure a file that has permits outstanding.

Attribute-word bits:

X'2000	Attribute Protected
X'1000	Permanent File
X'0800	Execution Sharable
X'0400	Read Sharable
X'0200	Write Sharable
X'0080	Secure File

X'0040 Read Only

X'0020 Write Only

X'0002 Top Loading

X'0001 Unsecure File (clears Secure File bit in directory)

X'C11C Prohibited Bits (representing attributes that cannot be changed by CHTR, e.g., Permits are outstanding); must be zero.

REWIND SEQUENTIAL FILE (RWND)

```
CALL
.WORD RWND
error return
normal return
```

Entry Parameters: AC2 = UFT address.

Exit Parameters: AC0 = Error code if error return.
AC1 = Changed.
AC2, AC3 = Unchanged.

Error Codes: ERFAP File attributes prohibit.

RWND sets the read pointers back to the beginning of a disk file, or rewinds a magnetic tape to its load point.

DISK SPACE AVAILABLE (DSKSP)

```
CALL
.WORD DSKSP
error return
normal return
```

Entry Parameters: AC2 = UFT address.

Exit Parameters: AC0 = Number of sectors available (unsigned), or error code on error return.

AC1 = Number of sectors used (unsigned), or undefined on error return.

AC2, AC3 = Unchanged.

Error Codes: ERBNA Buffer not available.
ERFND File name device prefix is not a disk.

DSKSP returns the number of sectors available and the number of sectors used on a disk.

READ DISK VOLUME IDENTIFICATION (VOLID)

```
CALL
.WORD VOLID
error return
normal return
```

Entry Parameters: AC0 = Address of File Name Block.
AC1 = Address of 12-word buffer.

A File Name Block prefix that is null specifies the nominal disk. A File Name Block prefix containing "$" specifies the system's primary disk. The root and extension may contain any values except the name of a device.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: AC0-AC3 = Unchanged.

(AC0 + 0, ... AC0 + 11) = Directory's volume identification entry for the media in the disk device specified.

Error Codes:

ERDIO	Disk I/O error.
ERDVNA	Device not available.
ERFDT	Device type prohibits.
ERFDSK	Root and extension name a disk.
ERFNDD	Root and extension name a non-disk device.
ERFND	Prefix does not specify a disk.

VOLID returns the whole volume identification entry from the directory on the media in the specified disk drive. This volid is specified to and written by FORMAT. Currently the meaningful words in this directory entry are:

Word	Symbol	Meaning
0 1 2	FCNM	File name root
3	FCNE	File name extension
5	FCDT	Date formatted

If the root and extension specify a disk, VOLID returns error code ERFDSK. If the root and extension specify a non-disk device, VOLID returns error code ERFNDD. If the prefix of the file name block does not specify a disk, then VOLID returns error code ERFND.

If the FNB does not specify any currently existing System File Table (SFT), then VOLID returns error code ERFNF. If the FNB specifies a non-disk device, then VOLID returns error code ERFDT. If the the specified disk's long term lock is set, then VOLID returns error code ERDVNA.

Chapter 8

HASH FILE MANAGEMENT

FILE CONCEPTS

System II hash file management provides record oriented direct and sequential access support for hash files on all System II disks. A hash file can only be accessed through the hash file management routines.

A hash file is organized to allow access to individual records based on a key field. Instead of reading through the file from the beginning to find a particular record, a mathematical computation is performed on the key field to produce the location of a node which contains the desired record or space to add a new record. Conceptually, a hash file consists of a number of contiguous nodes on a disk. Each node may be empty or may contain one or more records. The records at a node each have their own key field, the values of which may or may not be the same. The only relationship between records at a node is that their associated key fields all produced the same result when the hash calculation was performed.

A hash file must be initialized before it can be used. The initialization is performed as a part of the hash file creation process (see System II Utilities, V2-005, HCREAT). The creation process involves making a disk directory entry, allocating contiguous index space on the disk, marking all nodes as empty and storing the key field length, in words, into the directory entry. In addition, a hash file may be specified as allowing records with duplicate key fields or as a "batch" file (see Data-Rite Manual V1-057); no duplicate keys is the default.

FILE NAMES

File names are specified in the standard manner (see Chapter 2).

FILE ATTRIBUTES

Hash files are initialized with the following attributes:

- Hash file
- Shared for reading
- Shared for writing

Two optional attributes are:

- Duplicate keys permitted, or prohibited
- Batch file

A hash file's attributes can be changed (system call CHTR or utility CHATR) to add or drop any of the following attributes:

- AP (Attribute protected)
- PF (permanent file)
- SF (secure file)
- RD (read only)
- WO (write only)

Changing a hash file's attributes does not change any of the attributes listed in the initialization section above.

SHARING FILES

Hash files are created as sharable files. As with all disk files, the current availability for sharing can be limited by open requests. The effect of the options used with open is temporary, lasting only until the file is closed.

SYSTEM I/O CONCEPTS

CREATING A NEW HASH FILE

The directory entry for a hash file is created with the HCREAT command. The creation process involves making a disk directory entry, allocating contiguous index space on the disk (data space is allocated dynamically), marking the index empty and storing the key field length (in words) into the directory entry.

OPENING A HASH FILE

Prior to performing any file input or output, the user must acquire the file through a successful open request. An open will be refused if the hash file name is not present in the system directory or is not available for the use requested.

A hash file normally is opened with the OPEN call. The open request is processed in the same manner as that described for sequential files. OPNW or OPNR may also be used to open a hash file if limited I/O functions are desired.

RECORD ACCESS AND DATA TRANSFER

A hash file is accessed in terms of logical records rather than lines or blocks. Access may be "direct" or sequential. Direct access requires a key as input and uses the hash calculation on the binary value of the key to locate the proper record for update or retrieval. Sequential access is available only for reading the file. In this case, retrieval begins with the first record from the first node in the file and, with subsequent calls, proceeds to the next record in that node, and then the first record in the next node, until an end of file occurs.

The following calls are available:

- HFADD - Add a record.
- HFDEL - Delete a record.
- HFFND - Retrieve a record.
- HFNXT - Retrieve the next occurrence of the same key (only useful in files containing records with duplicate keys).
- HFXCH - Replace (exchange) a record.
- HLFND - Retrieve a record and lock it.
- HLNXT - Retrieve the next occurrence of the same key and lock the record.
- HREAD - Read sequentially.

CLOSING A HASH FILE

When all file accesses are complete, a CLOS call is used to disassociate the file from the job and free system resources used in file processing.

DELETING A HASH FILE

A hash file can be deleted or cleared using the command HDELT. There is a system call, DELHF, that can be used to delete a hash file in assembly programs. A hash file can be cleared but not deleted using the HFCLR call. The command DELETE will not delete a hash file. HFCLR is not a "record access or data transfer" call. qt requires that the target is closed; HFCLR is like HDELT.

RECORD LOCK

Record lock is provided by the hash file management routines Find and Lock Record (HLFND) and Find and Lock Next Record (HLNXT). They enable a user to prevent the record retrieved from being accessed by another user until the first user clears the lock by making another file access call through the same UFT. The lock is effective against HFDEL, HFXCH, HLFND and HLNXT calls specifying a UFT other than the one used in setting the lock.

HLFND and HLNXT are especially useful for retrieval of a record that is to be updated, for example, by means of an exchange call. Record lock must be employed if a second user could modify the file while it is being accessed. Failure to observe this requirement can result in destroying the logical structure of the disk. Cross-linked files, a destroyed directory, or a destroyed bit map are the most common symptoms of the destruction of the logical structure of a disk.

HREAD, HFFND, and HFNXT ignore the record locks and should not be used with one UFT while another UFT is doing locked record access or changing the contents of the hash file.

HFADD also ignores the record locks because a new record can't be locked. HFADD may be used in conjunction with locked record access.

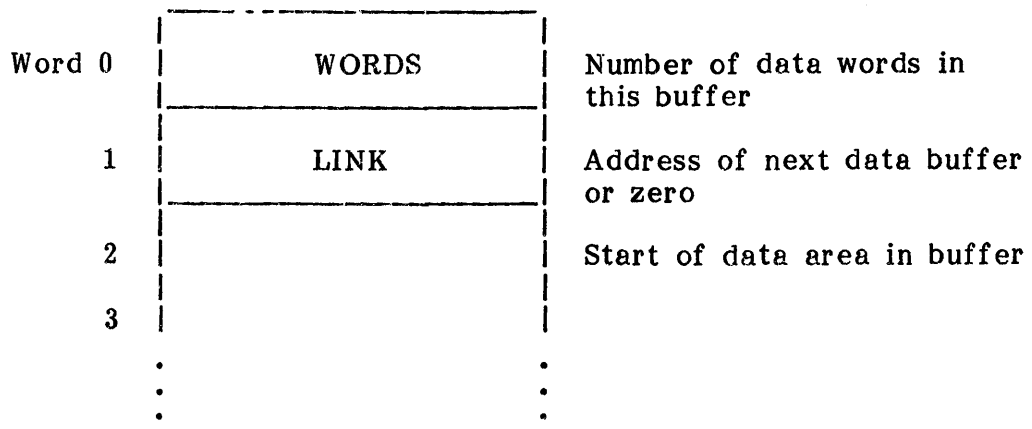
END-OF-FILE

HFNXT, HLNXT, HREAD give one end-of-file error and then start over at the beginning of the node or file.

HASH FILE FORMATS

RECORD BUFFER FORMAT

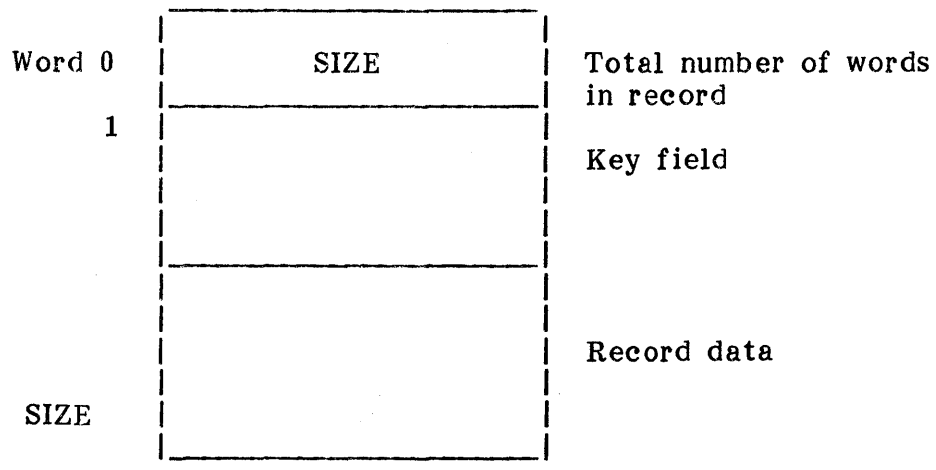
The hash file I/O routines require a specific format for both key field and record buffers. The data structure is designed to allow the use of linked system buffers, so large contiguous data areas are not required. Each buffer in the linked string must be in the following format:



Using the above format, a small record may be contained in one data area or system buffer while a large record may be placed into some number of linked system buffers.

RECORD FORMAT

System II hash file management allows the use of varying record lengths. The minimum numeric record size is the key field size specified when the hash file was created, while the maximum size allowed is 32,767 decimal. The record size is specified in the first word of the record. Each hash file record must be of the following form:



The key field must always be at the start of the record and is considered a part of the file record, as is the size specification. The key field is the only part of a record which must be a fixed length. The data following the key field may be of varying length as long as the total record size limitation is not exceeded. In the case of a file access or inquiry, when only the key field is required as input, the above format is required but SIZE is ignored.

HASH FILE ADD RECORD (HFADD)

```
CALL
.WORD HFADD
error return
normal return
```

Entry Parameters: AC1 = Address of first buffer in chain of buffers containing the record to be added to the hash file.

AC2 = Address of the file's UFT.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: AC0-AC3 = Unchanged.

Error Codes:

ERBNA	System buffer not available.
ERDIO	Disk I/O error.
ERDSNA	No disk space available.
ERFOP	Open options prohibit.
ERHDR	Duplicate record found.
ERHNHF	File not a hash file.
ERHRBE	Record buffers exceeded.
ERSCE	Hash file structural error.
ERVSP	Security prohibits.

HFADD adds the new record to the specified hash file. The record's key value is used by the hash algorithm to determine the node in which the record is to be placed. It is placed in the node in the first contiguous space large enough to contain it. HFADD considers a series of contiguous deleted records as one contiguous space. No program should be sensitive to the physical ordering of the records in a node.

If duplicate keys are not allowed in the specified hash file and a record with the same key value as that in the new record already exists, the hash file contents are not changed and HFADD returns error code ERHDR.

If the User File Table (UFT) does not allow writing (i.e., was obtained from OPNR), then HFADD returns error code ERFOP. If the file is secure and is not owned by the current security user who is not the SECOFF user, then HFADD returns error code ERVSP unless the current user had been granted add record permits at open time (PERMIT's /W option).

HASH FILE EXCHANGE RECORD (HFXCH)

```
CALL
.WORD HFXCH
error return
normal return
```

Entry Parameters: AC1 = Address of first buffer in chain of buffers containing a new record to replace the old.

AC2 = Address of UFT.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: AC0-AC3 = Unchanged.

Error Codes:

ERBNA	System buffer not available.
ERDIO	Disk I/O error.
ERDSNA	No disk space available.
ERFOP	Open options prohibit.
ERHBKL	Bad key length.
ERHNHF	File not a hash file.
ERHRBE	Record buffers exceeded.
ERHRNA	Record not available.
ERRNF	Record not found.
ERSCE	Hash file structural error.
ERVSP	Security prohibits.

HFXCH replaces an existing record with the specified record. HFXCH is equivalent to HFDEL immediately followed by HFADD.

If the hash file does not allow duplicate keys, then HFXCH replaces the only record that has the specified key.

If it allows duplicate keys, then HFXCH searches from the position indicated by current values in the UFT, and replaces the first record it encounters with the specified key. Hence, if the UFT is used with HFNXT to read the third record with a particular key, and then the UFT is used with HFXCH to replace a record with that key, then HFXCH replaces the third record.

In any case, the replacement is put in the first space large enough to contain it. This may be before or after the old record. No program should depend on the physical ordering of records in a node.

HFXCH returns error code ERRNF (record not found) if and only if there is no record in the specified hash file with the specified key. If the record to be replaced is locked by another User File Table (UFT), HFXCH returns error code ERHRNA. If the specified UFT does not allow writing (i.e., was obtained from OPNR), then HFXCH returns error code ERFOP. If the file is secure and is not owned by the current security user who is not the SECOFF user, then HFXCH returns error code ERVSP unless the current user had been granted modify access at open time (PERMIT's /M option).

HFXCH deletes the old record by marking it as deleted; it continues to reside in the same space. While HFADD and HFXCH may reuse the space occupied by deleted records, the quantity of such space usually grows as records are deleted and replaced. This costs execution time as well as disk space because the deleted records must be read in order to reach undeleted

records. Consequently, the hash file should be repacked periodically by creating a new hash file and using HKOPY to copy the old file to the new file.

HASH FILE FIND RECORD (HFFND)

```
CALL
.WORD HFFND
error return
normal return
```

Entry Parameters: AC1 = Address of first buffer in chain of buffers containing a record key.

AC2 = Address of UFT.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: AC0 = Address of first buffer in chain of record buffers.

AC1-AC3 = Unchanged.

Each record buffer must be freed by calling FBF.

Error Codes:

ERBNA	System buffer not available.
ERDIO	Disk I/O error.
ERFOP	Open options prohibit.
ERHBKL	Bad key length.
ERHNHF	File not a hash file.
ERRNF	Record not found.
ERSCE	Hash file structural error.

HFFND returns a copy of the first record with the specified key.

If the hash file does not allow duplicate keys, HFFND returns the unique record with the specified key. If the hash file allows duplicate keys, HFFND searches the node determined by hashing the specified key, and returns the first record it finds with that key. No program should be sensitive to the physical ordering of the records in a node. In particular, the record returned by HFFND may or may not be the record that was added first in time.

The record is returned in the shortest chain of 64-word system buffers sufficient to contain it. When processing of this record is finished, each of these buffers should be freed by calling FBF. If they are not explicitly freed, job termination frees them.

HFFND ignores record locks.

If the UFT does not allow reading (i.e., was obtained from OPNW:), then HFFND returns error code ERFOP.

If the file is secure and is not owned by the current security user who is not the SECOFF user, then the current user can open the file for read access only if the current user has been granted keyed read access (PERMIT's /F or/R option).

HASH FILE FIND NEXT RECORD (HFNXT)

```
CALL
.WORD HFNXT
error return
normal return
```

Entry Parameters: AC1 = Address of first buffer in chain of buffers containing record key.

AC2 = Address of UFT.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: AC0 = Address of first buffer in chain of record buffers.
AC1-AC3 = Unchanged.

Each record buffer must be freed by calling FBF.

Error Codes:

ERBNA	System buffer not available.
EREOF	End-of-file.
ERDIO	Disk I/O error.
ERFOP	Open options prohibit.
ERHBKL	Bad key length.
ERHNHF	File not a hash file.
ERRNF	Record not found.
ERSCE	Hash file structural error.

HFNXT searches the node determined by hashing the specified key, and returns a copy of the next record, with that key. The next record is the record physically next to the last one accessed. No program should depend on the physical ordering of records in a node. This ordering may have no resemblance to the chronological order in which the records were added to the file.

If the record at the position indicated by current values in the UFT does not have the specified key, the HFNXT acts just like HFFND and returns the first record with the specified key.

HFNXT is intended to be used only on hash files which permit duplicate keys, but it may be used on any hash file.

The record is returned in the shortest chain of 64-word system buffers sufficient to contain it. When processing of this record is finished, each of these buffers should be freed by calling FBF. If they are not explicitly freed, job termination frees them.

When the record at the position indicated by the UFT has the specified key, and no subsequent record has the same key, HFNXT returns error code EREOF (end-of-file). If HFNXT is called immediately after returning EREOF, HFNXT returns the first record with the specified key. HFNXT ignores record locks.

If the UFT does not allow reading (i.e., was obtained from OPNW), then HFNXT returns error code ERFOP.

If the file is secure and is not owned by the current security user who is not the SECOFF user, then the current user can open the file for read access only if the current user has been granted keyed read access (PERMIT's /F or /R option).

HASH FILE DELETE RECORD (HFDEL)

```
CALL
.WORD HFDEL
error return
normal return
```

Entry Parameters: AC1 = Address of first buffer in chain of buffers containing a record key.

AC2 = Address of UFT.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: AC0-AC3 = Unchanged.

Error Codes:

ERBNA	System buffer not available.
ERDIO	Disk I/O error.
ERFOP	Open options prohibit.
ERHBKL	Bad key length.
ERHNHF	File not a hash file.
ERHRNA	Record not available.
ERRNF	Record not found.
ERSCE	Hash file structural error.
ERVSP	Security prohibits.

HFDEL deletes one record from the specified hash file.

If the hash file does not allow duplicate keys, then HFDEL deletes the only record in the hash file that has the specified key.

If the hash file allows duplicate keys, then HFDEL deletes the first record with the specified key encountered when searching from the position indicated by current values in the UFT. Hence, if the UFT is used with HFNXT to read the third record with a particular key and then the UFT is used by HFDEL to delete a record with that key, then HFDEL deletes the third record, that is, the last obtained by HFNXT. If HFDEL is immediately called again, it searches for a fourth record with the specified key. If none is

found, then HFDEL continues the search from the beginning of the node. In this case, it would delete the (physically) first record with the specified key.

HFDEL returns error code ERRNF (record not found) if and only if there is no record in the specified hash file with the specified key. If the record to be deleted is locked by another User File Table (UFT), HFDEL returns error code ERHRNA.

If the specified UFT does not allow writing (i.e., was obtained from OPNR), then HFDEL returns error code ERFOP. If the file is secure and is not owned by the current security user who is not the SECOFF user, then HFDEL returns error code ERVSP unless the current user had been granted delete access at open time (PERMIT's /D option).

HFDEL deletes a record by marking it as deleted. The deleted record takes up the same space used by the record. While HFADD and HFXCH can reuse the space occupied by deleted records, the quantity of such space usually grows as records are deleted and replaced. This costs execution time as well as disk space because the deleted records must be read in order to reach undeleted records. Consequently, the hash file should be repacked periodically by creating a new hash file and using HKOPY to copy the old file to the new file.

HASH FILE SEQUENTIAL READ (HREAD)

```
CALL
.WORD HREAD
error return
normal return
```

Entry Parameters: AC2 = Address of UFT.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: AC0 = Address of first buffer in chain of record buffers.

AC1-AC3 = Unchanged.

Each record buffer must be freed by calling FBF.

Error Codes:

ERBNA	System buffer not available.
ERDIO	Disk I/O error.
EREOF	End of file encountered.
ERFOP	Open options prohibit.
ERHNHF	File not a hash file.
ERSCE	Hash file structural error.
ERHUP	UFT position lost due to changes made by other UFTs.
ERVSP	Security prohibits.

HREAD returns a copy of the next record in the specified hash file.

If the immediately preceding call using the specified User File Table (UFT) was not HREAD, then HREAD returns the first record in the hash file. If the immediately preceding call using the specified UFT was HREAD, then it returns the record following (physically) the record returned by the previous call in the sequential ordering of the hash file obtained by appending the second node to the first node, the third node to the second node, etc.

No program should depend on a simple relationship between key and node number. The hash algorithm is a pseudorandom number generator that evenly distributes the domain of keys over the range of nodes.

No program should depend on the sequential ordering of records within a node. This ordering may have no resemblance to the order in time in which the records were added to the file.

The record is returned in the shortest chain of 64-word system buffers sufficient to contain it. When processing of this record is finished, each of these buffers should be freed by calling FBF. If they are not explicitly freed, job termination frees them.

HREAD treats the hash file as a circular structure with the first node following the last node. If HREAD is called immediately after it returns an error code, HREAD returns the first record in the next node. This means that HREAD may be used to obtain all the recoverable data from a hash file which contains structural errors or sectors that cause disk errors. It also means that HREAD returns error code EREOF once at the end of the file before starting over at the beginning.

If HREAD is to use the current position of the specified UFT within a node, and the first word of the key of the record at this position is not equal to its contents when the last call to HREAD using this UFT returned this record, then HREAD returns error code ERHUP. HREAD ignores record locks and does not lock the record it returns. Hence, in the window between two calls to HREAD, a second UFT can be used to change the contents of the hash file in such a way that the first UFT is no longer positioned at the beginning of a record. This problem can be avoided by opening the hash file without write sharing.

If the UFT does not allow reading (i.e., was obtained from OPNW), then HREAD returns error code ERFOP. If the file is secure and is not owned by the current security user who is not the SECOFF user, then HREAD returns error code ERVSP unless the current user had been granted sequential read access at open time (PERMIT's /R option).

HASH FILE FIND AND LOCK RECORD (HLFND)

```
CALL
.WORD HLFND
error return
normal return
```

Entry Parameters: AC1 = Address of first buffer in chain of buffers containing a record key.

AC2 = Address of UFT.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: AC0 = Address of first buffer in chain of record buffers.

AC1-AC3 = Unchanged.

Each record buffer must be freed by calling FBF.

Error Codes:

ERBNA	System buffer not available.
ERDIO	Disk I/O error.
ERFOP	Open options prohibit.
ERHBKL	Bad key length.
ERHNHF	File not a hash file.
ERHRNA	Record not available.
ERRNF	Record not found.
ERSCE	Hash file structural error.

HLFND locks the first record in the specified hash file with the specified key and returns a copy of it.

If the hash file does not allow duplicate keys, HLFND returns the unique record with the specified key. If the hash file allows duplicate keys, HLFND searches the node determined by hashing the specified key, and returns the first record it finds having the specified key. No program should be sensitive to the physical ordering of the records in a node. In particular, the record returned by HLFND may or may not be the record that was added first in time.

Each User File Table (UFT) is allowed only one record lock. Each hash file call clears any record lock previously set by the specified UFT.

The record is returned in the shortest chain of 64-word system buffers sufficient to contain it. When processing of this record is finished, each of these buffers should be freed by calling FBF. If they are not explicitly freed, job termination frees them.

If the record to be locked is already locked by another User File Table (UFT), HLFND returns error code ERHRNA.

If HLFND returns error code ERHRNA, access to the locked record can be retried by calling HLFND again. The locked record can be skipped by calling HLNXT.

If the UFT does not allow reading (i.e., was obtained from OPNW), then HFFND returns error code ERFOP.

If the file is secure and is not owned by the current security user who is not the SECOFF user, then the current user can open the file for read access only if the current user has been granted keyed read access (PERMIT's /F or /R option).

HASH FILE FIND AND LOCK NEXT RECORD (HLNXT)

```
CALL
.WORD HLNXT
error return
normal return
```

Entry Parameters:
- AC1 = Address of first buffer in chain of buffers containing a record key.
- AC2 = Address of UFT.

Error Exit:
- AC0 = System error code.
- AC1-AC3 = Unchanged.

Normal Exit:
- AC0 = Address of first buffer in chain of record buffers.
- AC1-AC3 = Unchanged.

Each record buffer must be freed by calling FBF.

Error Codes:

ERBNA	System buffer not available.
ERDIO	Disk I/O error.
EREOF	End-of-file encountered.
ERFOP	Open options prohibit.
ERHBKL	Bad key length.
ERHNHF	File not a hash file.
ERHRNA	Record not available.
ERRNF	Record not found.
ERSCE	Hash file structural error.
ERVSP	Security prohibits.

HLNXT locks the next record with the specified key in the specified hash file and returns a copy of it. The next record is the physically next record in the same node. No program should depend on the sequential ordering of records in a node. This ordering may have no resemblance to the order in time in which the records were added to the file.

If the record at the position indicated by current values in the UFT does not have the specified key, then HLNXT acts just like HLFND by locking and returning a copy of the first record with the specified key.

HLNXT is intended to be used only on hash files which permit duplicate keys, but it may be used on any hash file.

Each User File Table (UFT) is allowed only one record lock. Each hash file call clears any record lock previously set by the specified UFT.

The record is returned in the shortest chain of 64-word system buffers sufficient to contain it. When processing of this record is finished, each of these buffers should be freed by calling FBF. If they are not explicitly freed, job termination frees them.

When the record at the current position of the specified UFT has the specified key and no subsequent record has the same key, HLNXT returns error code EREOF (end-of-file). If HLNXT is called immediately after returning EREOF, HLNXT returns the first record with the specified key.

If the record to be locked is already locked by another User File Table (UFT), HLNXT returns error code ERHRNA.

If HLNXT returns error code ERHRNA, the locked record may be retried by calling HLFND. The locked record may be skipped by calling HLNXT a second time.

If the UFT does not allow reading (i.e., was obtained from OPNW), then HLNXT returns error code ERFOP.

If the file is secure and is not owned by the current security user who is not the SECOFF user, then the current user can open the file for read access only if the current user has been granted keyed read access (PERMIT's /F or /R option).

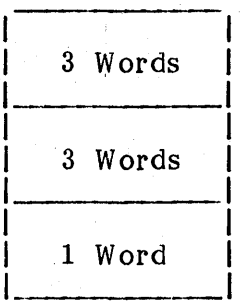
DELETE A HASH FILE (DELHF)

```
CALL
.WORD DELHF
error return
normal return
```

Entry Parameters:	AC0 = Address of file name block.	
Error Exit:	AC0 = Error code if error return. AC1-AC3 = Unchanged.	
Normal Exit:	AC0-AC3 = Unchanged.	
Error Codes:	ERBNA	Buffer not available.
	ERDIO	Disk I/O error.
	ERFAP	File attributes prohibit request.
	ERFNA	File not available now.

ERFNF	File name not found in directory.
ERVLG	Not logged on.
ERVOW	Only the owner may.
ERVPP	Permits outstanding.

File Name Block



See File Name Blocks, page 7-8, for formats of file name blocks.

DELHF deletes a hash file name from the file name directory and releases any disk space associated with that file. The file cannot be deleted while it is open. A sequential file cannot be deleted with this call.

If the file is open, then DELHF returns error code ERFNA. If the file is permanent (PF) or is not a hash file (not HF), then DELHF returns error code ERFAP. If the file is secure and the current user is not logged on, then DELHF returns error code ERVLG. If the file is secure and the current file security user is not the file's owner nor the SECOFF user, then DELHF returns error code ERVOW.

CLEAR A HASH FILE (HFCLR)

```
CALL
.WORD HFCLR
error return
normal return
```

Entry Parameters: AC0 = Address of file name block.

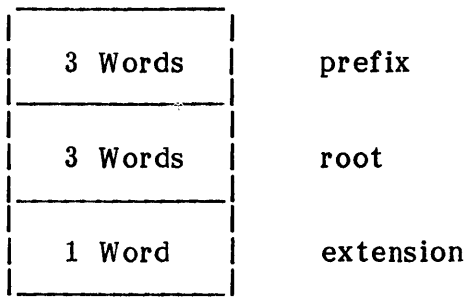
Error Exit: AC0 = Error code if error return.
AC1-AC3 = Unchanged.

Normal Exit: AC0-AC3 = Unchanged.

Error Codes:

ERBNA	Buffer not available.
ERDIO	Disk I/O error.
ERFAP	File attributes prohibit request.
ERFNA	File not available now.
ERFNF	File name not found in directory.
ERVLG	Not logged on.
ERVOW	Only the owner may.

File Name Block



See File Name Blocks, page 7-8, for formats of file name blocks.

HFCLR releases all of the hash file's data sectors and reinitializes its index sectors. Only its owner is allowed to clear a secure hash file.

HFCLR stops at the first error. Hence, HFCLR may leave the hash file partially cleared.

If the file is open, then HFCLR returns error code ERFNA. If the file is not a hash file (not HF), then HFCLR returns error code ERFAP. If the file is secure and the current user is not logged on, then HFCLR returns error code ERVLG. If the file is secure and the current file security user is not the file's owner nor the SECOFF user, then HFCLR returns error code ERVOW.

(THIS PAGE INTENTIONALLY BLANK)

Chapter 9

COMPUTER TERMINALS

Jacquard computers include one or more Cathode Ray Tube (CRT) terminals, each with a keyboard. The terminals may be either Standard or International; some documentation refers to the latter as the Universal terminal. Different character sets are involved: a fixed set for the Standard terminal, and any of a variety of optional sets for the International terminal. Furthermore, the J500 allows user-defined dot patterns for characters in the range X'80 to X'FF. For details, refer to the Utilities Manual under CHRGEN.

Standard and International terminals both have a 1920-character screen 80 columns wide by 24 lines long. The display is produced from character codes stored in a refresh memory. This memory is accessible to both the refresh generator and the CPU. Data is output to the screen by storing characters in the refresh memory.

A separate 1024-word area of memory must be assigned to each Standard terminal in the system. For International terminals, all users share the same 1024-word area.

Various screen features, such as blinking and underscoring, are controlled by the display of certain codes. Details are given at the end of this section.

When a terminal I/O operation expects keyboard input, a cursor (a blinking underscore) appears at the relevant screen position. At other times, when no input is expected, the cursor disappears.

Computer keyboards include character keys, shift keys, edit keys, and function keys. Character keys generate data according to the current shift. The standard keyboard has one shift for uppercase. The International keyboard has three shift modes, with effects which depend on the options ordered. For example, on the English-language version, <SHIFT 1> operates like a typewriter shift. <SHIFT 2>, used alone or in combination with <SHIFT 1>, generates additional characters.

Edit keys cause System II to perform some control function; cursor movement, character insertion, line deletion, and so on, as detailed later in this section.

Function keys (<F1> to <F10> on the Standard keyboard, <F1> to <F20> on the International keyboard) are programmable. When the CRT is in Free Screen or Split Screen Mode, their meanings are defined by the user program.

Details on keyboard code generation and processing appear later in this section.

TERMINAL INPUT/OUTPUT HANDLER

The terminal I/O handler is a section of re-entrant software in the operating system which can handle any number of terminals simultaneously. Its basic functions are to:

- Process interrupts from the keyboard and display keyed characters on the screen.
- Position the cursor on the screen as determined by keyboard inputs and system calls.
- Obtain characters from the refresh memory in response to read requests.
- Place characters in the refresh memory in response to write requests.
- Create standard error messages in response to requests from programs.
- Accept command lines entered by the operator.

At any given time a terminal is in one of four states:

Command Mode
Roll Mode
Free Screen Mode
Split Screen Mode

Command Mode enables the user to communicate with the operating system by entering command lines. In this mode no job is attached to the terminal. In the other three modes, a particular program is attached to the terminal, and can open it. Only the program which has opened the terminal can communicate with it.

COMMAND MODE

In Command Mode, the terminal accepts keyboard input at any time. Characters typed on the keyboard are displayed on the bottom line of the screen. As each character is typed, it is displayed at the position indicated by the cursor, and the cursor is advanced one position to the right.

<RETURN> is pressed to indicate keyboard input is finished and the line is ready to be processed by the system. After the key is struck, the screen is rolled up, the bottom line is cleared, the cursor position is set to the left end of the bottom line and the cursor is removed from the display.

If the system recognizes the line as a valid command, the terminal becomes the message terminal for the job initiated by the command. It is taken out of Command Mode and left in a state where it can be opened by the program. The terminal will be in Roll Mode after it is opened. When it is subsequently closed, it will be left in a state where it can be reopened by the program. It will not be returned to Command Mode until the program terminates itself or releases the terminal.

If the system does not recognize the command, an error message is displayed, the screen is rolled up, the bottom line is cleared and the cursor is displayed at the left end of the bottom line - indicating that the terminal is ready to accept another command. <DOWN ARROW> can be used to roll the incorrect command line down to the bottom line for correction.

If an application program releases the terminal, the terminal is returned to Command Mode. However, it is still considered the message terminal for the job unless the job subsequently opens another terminal.

Any program can open a terminal that is in Command Mode if the bottom line is blank. In that event, the cursor is removed from the display and the terminal is taken out of Command Mode and put into Roll Mode. When it is subsequently closed, it will be put back into Command Mode unless it is the job's message terminal.

ROLL MODE

Roll Mode is so named because, in this state, a string that is output to the terminal is displayed on the bottom line, and when the string ends, the screen is rolled up (scrolled upwards one line) and the bottom line is cleared, making it ready for a new line. A terminal defaults to Roll Mode on being opened. Roll Mode can also be entered by means of the ROLCRT call.

In Roll Mode, output can be transferred to the screen in strings of user-specified length (WRS call) or delineated by an explicit line terminator (WRL, WCRTB calls). Characters are transferred to the screen beginning at the current cursor position. After each transfer, the cursor is set one column to the right of the last character transferred to the screen. Whenever a line terminator is encountered, the screen is rolled up, the bottom line is blanked and the cursor is set to the left end of the bottom line.

Input from the keyboard to the bottom line of the screen can be made only if the application program issues a request to read that line by an RDL, RCRTB or RCRTBN call. <RETURN> is pressed to indicate that keyboard input is finished. At that time, the screen is rolled up, the bottom line is cleared, the cursor position is set to the left end of the bottom line and

the cursor is removed from the display. A special read request (RCRTBN call) is available to suppress the roll after the read is finished.

When a terminal read is done, the cursor is displayed at the current cursor position as determined by the previous output call. Thus, if the last output call was a WRS, the cursor may appear anywhere on the bottom line. If the last call was WRL, WCRTB or RDL the cursor will be displayed at the left end of the bottom line, which will be blank.

FREE SCREEN MODE

Free Screen Mode enables a program to display or accept characters from any place on the screen. This mode is entered by means of the FULCRT call.

In this mode, fields of data are transferred to the screen by means of a WCRT call. Each call must supply a character string, its length, and the X-Y coordinates (column, line) on the screen where the string is to begin.

Input from the keyboard is possible only if the application program issues a read request (RCRT) to the terminal, that is, a request to read keyboard input to the screen. In response to a read, the cursor is displayed at the X-Y coordinates specified by the RCRT call. The cursor should be positioned in an unprotected field. Protected fields cannot be overwritten by keyboard input. A field is protected if it begins with a protect code (X'1E) and ends with an unprotect code (X'1F). As each character is typed, it is displayed at the position indicated by the cursor, and the cursor is advanced one position to the right. If that position contains a protect code, the cursor is skipped right to the beginning of the next unprotected field. This prevents the keyboard operator from overwriting protected fields.

Keyboard input is terminated when any function key (<F1>-<F20>) is pressed. The function key code is returned to the program which issued the terminal read. After the read is finished, the program can use the MCRT call to read fields from the screen and write them into a user work area.

SPLIT SCREEN MODE

Split Screen Mode is a combination mode in which the top of the screen is in Free Screen Mode and the bottom is in Roll Mode. Split Screen Mode is entered by a SPCRT call and is cleared by either a NSPCRT call or termination of the program. By default the top 15 lines are in free mode and the bottom 9 are in roll mode. A SPBCRT call can be used to change where the split occurs.

KEYBOARD TRANSLATION

The calls RCRT, RCRTB, RCRTBN, RDL, and RDS allow keyboard input. If there is a keyboard translation table, each 8-bit key code is translated before it is stored into the terminal refresh. When RCRTB, RCRTBN, RDL, or RDS terminates, the translated key code is moved from the terminal refresh

to the user's buffer. After RCRT terminates, MCRT can be used to copy the translated key codes from the terminal refresh to the user's buffer.

The TCUP utility program installs or frees a keyboard translation table.

Keyboard Translation Table Format

A translation table is 256 words long. These words are numbered 0, 1, 2, ... 255. Word n contains the translations for the 8-bit byte with value n. If lowercase is not allowed, the left byte of word n is the translation of n. If lowercase is allowed, the right byte of word n is the translation of n.

TERMINAL MESSAGE FACILITY

The operating system provides a facility for displaying messages on a terminal. These messages may be system error messages or user specific, and the message text may be held in a disk file. Such message files are created and maintained by the MSGUPD utility which is described in the appropriate section of the System II Utilities Manual, V2-005.

SYSTEM MESSAGE LINE

The message routines will display the message on the system message line. Note that MSRD will optionally display a message at any given screen position.

The system message line is either the bottom line or the next to the bottom line on the screen. Use of the system message line involves either rolling the roll part of the screen or blanking the message line before displaying the message.

If the terminal is in command mode or has a roll mode read in progress, the system message line is the next to the bottom line. The roll part of the screen down to the next to the bottom line is rolled up one line before the message is displayed on the next to the bottom line. Note that this does not change the contents of the bottom line.

If the terminal is not in command mode or full mode, and does not have a roll mode read in progress, the system message line is the bottom line. The roll part of the screen is rolled up one line before the message is displayed on the bottom line.

If the terminal is in full mode, the system message line is the bottom line. The bottom line is blanked before the message is displayed.

SYSTEM ERROR MESSAGE FILE

Any message file may be declared to be the System Error Message File by using the MSGFIL utility. If such a system error message file has been

declared, then the appropriate text will be retrieved from this file and will appear on all error messages displayed by the system. System II provides a standard error message file called SY2ERF.

The more general message file facility is used via the system call MSRD. There may be any number of message files, but there can only be one system error message file. Note that MSRD optionally uses the system error message file if so requested.

SYSTEM ERROR MESSAGE FORMAT

The default formats for system error messages are documented in the Error Messages Manual, V3-005. The following formats are used when there is a system error message file and no error occurs in accessing it.

```
*program [dsknam A=aaaa H=hhhh] [FN=dddddd:ffffff.ff] message text
```

program is the current job's name (from the job name buffer). If the current job is the operating system, this name will be "(Cx) SY Rxx.x", where Cx indicates the configuration and Rxx.x indicates the release level. This field is always present and always contains 17 bytes (including trailing blank).

[dsknam . . .] appears only on a system error 8 (disk error). Dsknam is the name of the disk on which the error occurred. aaaa is the logical disk address in hexadecimal on which the error occurred. hhhh is the hardware status in hexadecimal. When this field appears it contains 21 bytes or less (including trailing blank).

[FN=...] appears only if FMSG or MSVF was called. Dddddd:ffffff.ff is the name of the file associated with the UFT specified to FMSG or MSVF. When this field appears it contains 20 bytes or less (including trailing blank).

Hence the message text for system error 8 to be used via FMSG or MSVF should not contain more than 22 bytes. Message text for a system error other than 8 to be used via FMSG or MSVF should not contain more than 43 bytes.

message text for system error 8 to be used only via SMSG or MSVS should not contain more than 42 bytes. Message text for a system error other than 8 to be used only via SMSG or MSVS should not contain more than 63 bytes.

Message text that is too long is truncated when it is displayed.

Examples

1. Delete utility calls SMSG to report system error 34.

   ```
   *DELETE File not found
   ```

2. Delete utility calls SMSG to report system error 8.

   ```
   *DELETE FP00 A=0005 H=0020 Disk I/O error
   ```

3. Program DRLA calls FMSG to report system error 87.

   *DRLA FN=FP00:TEMP Record not found

4. Program DRLA calls FMSG to report system error 8.

   *DRLA FP00 A=0010 H=0060 FN=FP00:TEMP Disk I/O error

5. Operating system calls SMSG to report system error 7.

   (C1) SY R09.0 Command line error

TERMINAL CALLS

The various terminal calls are summarized below.

MODE SETTING CALLS

ROLCRT, NSPCRT	Set Roll Mode
FULCRT	Set Free Screen Mode
SPCRT	Set Split Screen Mode
CTLCRT,RELJ	Release Job (return terminal to Command Mode)

READ CALLS

RDL, RCRTB, RDS	Read Bottom Line (Roll Mode)
RCRTBN	Read Bottom Line, No Roll (Roll Mode)
RCRT	Read Screen (Free Screen Mode)
MCRT	Copy Data From Screen (Free Screen Mode)

WRITE CALLS

WCRT	Write Screen (Free Mode)
WCRTB	Write to Bottom Line (Free Mode or Roll Mode)
WRL	Write Line (Free Mode or Roll Mode)
WRS	Write Sequential (Free Mode or Roll Mode)

MESSAGE DISPLAY CALLS

SMSG	Display System Error Message
MSVS	Display System Error Message to any designated terminal
FMSG	Display System Error Message and File Name
MSVF	Display System Error Message and File Name to designated terminal
MSG	Display Message
MSRD	Message Read and Display

MISCELLANEOUS TERMINAL CALLS

CRTLCI	Set or Clear Lowercase Input Option
CRLGTS	Set or Read Terminal Status Lights
CEFREE	Erase Free Screen
CEPROT	Erase Only Unprotected Fields
CERALL	Erase Entire Screen Including Protected Fields
CEROLL	Erase Roll Part
SPBCRT	Set Split-Screen Boundary

SET ROLL MODE (ROLCRT, NSPCRT)

```
CALL                    CALL
.WORD ROLCRT            .WORD NSPCRT
error return            error return
normal return           normal return
```

Entry: AC2 = Address of UFT.

Error Return: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Return: Registers unchanged.

Error Codes: ERCRNC CT type call on a non-CT device.

ROLCRT and NSPCRT are exactly the same. They put the terminal into Roll Mode. The terminal is in Roll Mode by default when it is opened.

When ROLCRT (NSPCRT) mode clears Free Mode, the screen is rolled immediately before the next RDL, RCRTB, RCRTBN, WRS, WRL, or WCRTB call. This is in addition to any roll that may occur after these calls.

The cursor is left in the roll home position (lower left corner of screen).

For more details see the description of Roll Mode.

SET FREE SCREEN MODE (FULCRT)

```
CALL
.WORD FULCRT
error return
normal return
```

Entry: AC2 = Address of UFT.

Error Return: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Return: Registers unchanged.

Error Codes: ERCRNC CT type CALL on a non-CT device.

FULCRT puts the terminal into Free Mode. It does not change the cursor position. For more details see the description of Free Mode.

SET SPLIT SCREEN MODE (SPCRT)

```
CALL
.WORD SPCRT
error return
normal return
```

Entry: AC2 = Address of UFT.

Error Return: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Return: Registers unchanged.

Error Codes: ERCRNC CT type CALL on a non-CT device.

SPCRT puts the terminal into Split-Screen Mode. The default split screen boundary is 15. The split screen boundary may be changed to any value of K from 1 to 23 by calling SPBCRT. When the boundary is line K, lines 0, 1 ... K-1 are the Free Mode part; and lines K, K+1 ... 23, are the Roll Mode part. The cursor is left at the roll mode home position (lower left corner of screen).

SET SPLIT SCREEN BOUNDARY (SPBCRT)

```
CALL
.WORD SPBCRT
error return
normal return
```

Entry Parameters: AC0 = Number of lines to be in Free Screen mode.
AC2 = Terminal UFT address from OPEN.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: Registers unchanged.

Error Codes:

ERCRXY	Number of lines specified was outside the allowed range of 1 to 23.
ERCRNC	VCT type CALL on non-VCT device

The SPBCRT call sets the line number at which the split-screen boundary occurs. Specifying k as the number of lines causes lines 0 through k-1 to be in Free Screen mode, and lines k through 23 to be in Roll mode. Until this call has been made, the split occurs at line 15. When the terminal is closed, the split is reset to line 15.

This call may be made in any screen mode. Changing screen mode does not change the split-screen boundary.

RETURN TO COMMAND MODE (CTLCRT)

```
CALL
.WORD CTLCRT
error return
normal return
```

Entry Parameters: None.

Exit Parameters: AC0 = Error code if error.

Error Codes: ERCROP Terminal is open.

For details of CTLCRT's operation, see the description of RELJ.

READ BOTTOM LINE (RCRTB, RDL, RDS)

```
CALL                CALL              CALL
.WORD RCRTB or      .WORD RDL or      .WORD RDS
error return        error return      error return
normal return       normal return     normal return
```

Entry: AC0 = Address of user's line buffer.
AC1 = Maximum number of bytes to be read.
AC2 = Address of UFT.

Error Return: AC0 = System error code

AC1 = Number of bytes moved into the user's buffer including the line terminator (unless error ERLTL).

AC2-AC3 = Unchanged.

Normal Return: AC0, AC2-AC3 = Unchanged.
AC1 = Number of bytes read (including line terminator).

Error Codes:

ERCRNC	CT type call on a non-CT device.
ERLTL	Line too long.
ERABT	Keyboard abort.

RCRTB, RDL, and RDS read the bottom line of the terminal. They are all precisely the same. If the terminal is in free mode when any of these calls are made, then they put the terminal into roll mode. In some cases the terminal is left in roll mode when the error return is taken. This happens for example when the read is terminated by hitting the cancel key. If the terminal is in split mode when any of these calls are made, the mode is not changed.

The task is suspended until the read is completed.

If the current cursor position is not on the bottom line when this call is made, then it is set to the lower left corner of the screen. Then the cursor is displayed at the cursor position and input is allowed.

After the line has been copied to the user's work area, the screen is rolled up, the bottom line is blanked and the cursor position is set to the lower left corner of the screen. The cursor is no longer displayed and keyboard input is inhibited until the next read request.

READ BOTTOM LINE WITHOUT ROLL (RCRTBN)

```
CALL
.WORD RCRTBN
error return
normal return
```

Entry: AC0 = Address of user's buffer.
AC1 = Maximum number of bytes to read.

Error Return: AC0 = System error code.

AC1 = Number of bytes moved into the user's buffer including the line terminator (unless error ERLTL).

AC2-AC3 = Unchanged.

Normal Return: AC0, AC2-AC3 = Unchanged.
AC1 = Number of bytes moved into the user's buffer including the line terminator.

Error Codes:

ERABT	Keyboard abort.
ERCRNC	CT type call on a non-CT device.
ERLTL	Line too long.

RCRTBN is like RCRTB, RDL, and RDS except: (1) The screen is never rolled before input is allowed. (2) When the read is completed, the screen is not rolled. (3) The cursor is left at its position at the time the read was completed.

RCRTBN allows the application program to issue a second RCRTBN to allow corrections and additions to the line. Note that the entire non-blank contents of the line are copied to the user's buffer on each read.

After an RCRTBN call the screen is rolled immediately before processing the next RCRTB, RDL, RDS, WCRTB, WRL, or WRS call to the terminal.

READ FREE SCREEN (RCRT)

```
CALL
.WORD RCRT
error return
normal return
```

Entry: AC0 = X-coordinate (column position, 0 leftmost).
AC1 = Y-coordinate (line position, 0 topmost).
AC2 = Address of UFT.

Error Return: AC0 = System error code.
AC1-AC3 = Unchanged.

In some cases the terminal is changed from roll to free mode before the error return is taken.

Normal Return: AC0 = Function key code.
AC1-AC3 = Unchanged.

Error Codes:

ERCRNC	CT type call on a non-CT device.
ERCRXY	Cursor position specified lies in the roll part of the screen.
ERCRPR	A protect or unprotect code is at the cursor position specified.

RCRT reads the free part of the screen in Free or Split Mode. If the terminal is in Roll Mode when this call is made, then RCRT changes the mode

to Free. The task making this call is suspended until a keyboard response completes the read.

Unlike all the other terminal read calls, RCRT takes its normal return when the cancel key is used to terminate the read. In this case, just as in every other case, the cancel key sets the job abort flag and initiates abnormal termination of all the job's tasks.

An unprotected field that wraps around from the lower, right corner of the screen to the upper, left corner of the screen begins after the unprotect code, not at the upper left corner of the screen.

When the cursor position specified contains a protect or unprotect code, then every key is read terminating. This means that the first key depressed terminates the read, the read does not change the contents of the screen, and the key code is returned to the user. This allows the application program to handle every key.

If cursor motion can not be completed because the whole screen is protected, then RCRT takes its normal return.

COPY DATA FROM SCREEN (MCRT)

```
CALL
.WORD MCRT
error return
normal return
```

Entry:	AC0 = Address of transfer table. AC2 = Address of UFT.
Error Return:	AC0 = System error code. AC1-AC3 = Unchanged.
Normal Return:	AC0, AC2-AC3 = Unchanged. AC1 = Number of bytes moved from last field.
Error Codes:	ERCRNC CT type call on non-CT device. ERCRXY X or Y out of range.

MCRT copies data from the screen to user buffers. This call may be made in Free, Split, or Roll modes. It is usually used only on the free part of the screen in conjunction with RCRT.

MCRT's transfer table has the same structure as RCRT's transfer table (see RCRT). The bytes specified by the table are copied from the screen without regard for protected fields or other logical structure. The bytes are copied to the user's buffer two 8-bit bytes per 16-bit word in left-right order.

If a field begins on the screen but extends beyond the screen, the transfer stops at the end of the screen.

Transfer Table Structure

The transfer table must contain one four-word entry for each field of the screen to be copied. An entry gives the starting coordinates of the field, the length of the string, and the address of the work area into which it is to be written. Each entry in the table has the following structure:

```
Word 0 = Number of bytes to copy.
Word 1 = User work area address where data is to be written.
Word 2 = X-coordinate (column number) of field.
Word 3 = Y-coordinate (line number) of field.
```

The table must be terminated by an entry in which word 0 contains a zero.

Bytes in the specified strings are copied from the screen regardless of the logical structure of the display, e.g., without regard to whether a field is protected or not. Strings are stored in ASCII code (eight bytes per character), and written from left to right, two bytes per word, in the designated work area.

Note that the location having zero coordinates (X=0, Y=0) is the leftmost column of the top line of the screen.

WRITE TO BOTTOM LINE OF TERMINAL (WCRTB)

```
CALL
.WORD WCRTB
error return
normal return
```

Entry:	AC0 = Address of user's buffer. AC1 = Maximum number of bytes. AC2 = Address of UFT.
Error Return:	AC0 = System error code. AC1-AC3 = Unchanged.
Normal Return:	AC0, AC2-AC3 = Unchanged. AC1 = Number of bytes moved.
Error Codes:	ERCRNC CT type call on non-CT device. ERISZ Illegal size request.

WCRTB writes to the bottom line of a computer terminal. If when the call is made the cursor is not somewhere on the bottom line, it is set to the lower left corner of the screen. Each character is copied to the current cursor position and the cursor position is updated. If the cursor is at the right edge of the bottom line, the character is counted but is not copied and the cursor position is not changed. A line terminator (any code less than X'10 except X'09) is not put on the screen. A line terminator stops WCRCTB. The maximum byte count specified also stops WCRTB. In free mode the screen is never rolled, but the bottom line is blanked and the cursor set to the lower, left corner of the screen before copying any characters to the

screen, and the undisplayed cursor is left at the end of the string copied. In roll mode and split mode WCRTB rolls the screen whenever the bottom line is non-blank after the transfer. If the transfer does not end with a line terminator, this roll is included in the character count returned. The effect of this is that WCRTB never leaves the bottom line non-blank. The cursor is not displayed.

The count returned includes one for each tab expanded.

WRITE LINE TO TERMINAL (WRL)

```
CALL
.WORD WRL
error return
normal return
```

Entry: AC0 = Address of user's buffer.
AC2 = Address of UFT.

Error Return: AC0 = System error code.
AC1-AC3 = Unchanged.

This error return should never be used in normal operation. It indicates that the system has been damaged and should be re-booted as soon as possible.

Normal Return: AC0, AC2-AC3 = Unchanged.
AC1 = Number of bytes moved.

Error Codes: ERCRNC CT type call on a non-CT device

WRL with a computer terminal User File Table (UFT) writes to the bottom line of a computer terminal. If when the call is made the cursor is not somewhere on the bottom line, it is set to the lower left corner of the screen. Each character is copied to the current cursor position and the cursor position is updated. If the cursor is at the right edge of the bottom line, the character is counted but is not copied and the cursor position is not changed. A line terminator (any code less than X'10 except X'09) is required to stop WRL. The line terminator is not put on the screen. In free mode WRL never rolls the screen, but the bottom line is blanked and the cursor is set to the lower, left corner of the screen before copying any characters to it; the undisplayed cursor is left at the end of the string transferred. In roll mode and split mode, WRL ends by rolling the screen. This blanks the bottom line and leaves the undisplayed cursor at the lower, left corner of the screen. The cursor is not displayed.

The count returned includes one for each tab expanded.

WRITE SEQUENTIAL TO TERMINAL (WRS)

```
CALL
.WORD WRS
error return
normal return
```

Entry: AC0 = Address of user's buffer.
AC1 = Number of bytes.
AC2 = Address of UFT.

Error Return: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Return: AC0, AC2-AC3 = Unchanged.
AC1 = Number of bytes moved.

Error Codes: ERISZ Illegal size request

WRS with a computer terminal User File Table (UFT) writes to the bottom line of a computer terminal. If when the call is made the cursor is not somewhere on the bottom line, it is set to the lower, left corner of the screen. Each character is copied to the current cursor position and the cursor position is updated. If the cursor is at the right edge of the bottom line, the character is counted but is not copied and the cursor position is not changed. A line terminator (any code less than X'10 except X'09) is not put on the screen. In free mode, a line terminator stops WRS. Hence, in free mode WRS never rolls the screen, and the byte count returned may be smaller than the byte count specified. In roll mode and split mode, a line terminator causes the screen to roll. Hence, WRS rolls the screen only when it encounters a line terminator and the screen is in roll mode or split mode.

Tabs are expanded. The count returned includes one for each tab expanded and one for each line terminator encountered.

WRITE SCREEN (WCRT)

```
CALL
.WORD WCRT
error return
normal return
```

Entry: AC0 = Address of transfer table.
AC2 = Address of UFT.

Error Return: AC0 = System error code.
AC1, AC2-AC3 = Unchanged.

Normal Return: AC0, AC2-AC3 = Unchanged.
AC1 = Number of bytes moved to last field.

Error Codes: ERCRNC CT type call on a non-CT device.
ERCRXY Cursor position specified out of range.

WCRT copies data from a user's buffer onto the terminal screen. The screen may be in Roll, Free or Split mode.

WCRT is given the address of a transfer table specifying where the data is to be obtained and where it is to be copied on the screen. This table may contain any number of entries. Each entry is four words long and described below.

Copying each string of data will terminate either on the byte count specified or by encountering a line terminator. The byte count returned will include one for the terminator (if any). A line terminator is any value less than X'10 except for tab (X'09) which is expanded to blanks and X'01 which is stored on the screen. WCRT is the only way to write X'01 (the solid box character) on the screen. The byte count returned will include one for each tab expanded in the last field.

If a field begins within the screen and extends beyond the screen, all characters to be copied beyond the screen are counted but are not copied.

WCRT does not change the cursor position.

Transfer Table Structure

The transfer table contains a separate entry for each field to be copied. Each entry is four words long with the following structure:

Word 0 = Number of bytes to copy.

Word 1 = Address from which data is to be copied.

Word 2 = X-coordinate (column number) of field. (Byte offset into refresh if word 3 is zero.)

Word 3 = Y-coordinate (line number) of field.

The table must be terminated by an entry in which word 0 contains zero.

Note that X-Y coordinates (0, 0) specify the upper left corner of the screen.

DISPLAY SYSTEM ERROR MESSAGE (SMSG)

```
CALL
.WORD SMSG
error return
normal return
```

Entry: AC0 = System error code.

Error Exit: AC0 = System error code. Possibly different from the system error code input.

AC1-AC3 = Unchanged.

Normal Exit: AC0-AC3 = Unchanged.

Error Codes:
ERDIO Disk I/O error.
ERSCE Hash file structural error.
ERRNF Record not found.

SMSG displays on the current job's message terminal an error message for the specified system error code.

If messages are suppressed for this terminal, then SMSG does not display anything. If the current job is running in the background and has a predecessor, then SMSG does not display anything.

SMSG sets the current job's error code to the specified error code. The job's error code is passed back to the predecessor job when its execution is restarted at LINK's normal return.

Message

If there is currently no system error message file, the message displayed has the standard format described in the Error Messages manual. If there is a system error message file, the message has the following format:

*program [dsknam A=aaaa H=hhhh] message text

Fields have variable length as appropriate. Unused fields are not present. Those fields that are present are separated by one blank. Hence, the total message length is minimized.

program is the current job's name. If the current job is the operating system, this name will be "(Cx) SY Rxx.x", where Cx indicates the configuration and Rxx.x indicates the release level. If there is no current job, this field is not present.

[dsknam . . .] appears only on a system error 8 (disk error). Dsknam is the name of the disk on which the error occurred. Aaaa is the logical disk address in hexadecimal on which the error occurred. Hhhh is the hardware status in hexadecimal.

The message is displayed on the system message line in the manner described in the System Message Line section. If necessary, the message is truncated. Truncation is not an error and does not cause SMSG to take its error return.

Errors

If there is currently a system error message file and an error occurs in trying to access this file, SMSG returns the appropriate error code after displaying the standard format message for entry parameter AC0. Hence, if there is a system error message file, but it does not contain a message for the specified error code, SMSG displays the standard format error message for the error code and then returns error code ERRNF (record not found). Even when SMSG returns an error code, it sets the current job's error code to its entry parameter AC0.

ERROR MESSAGE TO COMPUTER TERMINAL (MSVS)

```
        CALL
        .WORD MSVS
        error return
        normal return
```

Entry: AC0 = System error code.

AC1 = Address of File Name Block specifying a computer terminal.

0 ---> display message on the current job's message terminal.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: AC0-AC3 = Unchanged.

Error Codes:

ERFNF	File not found.
ERCRNO	Terminal not a computer terminal.
ERDIO	Disk I/O error.

MSVS displays on the specified computer terminal an error message for the specified error code.

MSVS sets the current job's error code to the specified error code. The job's error code is passed back to the predecessor job when its execution is restarted at LINK's normal return.

Message

If there is currently no system error message file, the message displayed has the standard format described in the Error Messages manual. If there is a system error message file, the message has the following format:

*program [dsknam A=aaaa H=hhhh] message text

program is the current job's name (from the job name buffer). If the current job is the operating system, this name will be "(Cx) SY Rxx.x",

where Cx indicates the configuration and Rxx.x indicates the release level. This field is always present and always contains 17 bytes or less (including trailing blank).

[dsknam . . .] appears only on a system error 8 (disk error). Dsknam is the name of the disk on which the error occurred. Aaaa is the logical disk address in hexadecimal on which the error occurred. Hhhh is the hardware status in hexadecimal. When this field appears it contains 21 bytes or less (including trailing blank).

The message is displayed on the system message line in the manner described in the System Message Line section.

Errors

If the specified File Name Block does not specify an exising file, MSVS returns error code ERFNF. If the specified File Name Block specifies an existing file that is not a computer terminal, MSVS returns error code ERCRNO. If there is an error in accessing the system error message file, MSVS displays the standard format system error message for the specified error code before taking its error return. Even when MSVS takes its error return, it sets the current job's error code to its entry parameter AC0.

DISPLAY SYSTEM ERROR MESSAGE AND FILE NAME (FMSG)

```
        CALL
        .WORD FMSG
        error return
        normal return
```

Entry:	AC0 = System error code AC2 = UFT address of file associated with error.
Error Exit:	AC0 = System error code AC1-AC3 = Unchanged.
Normal Exit:	AC0-AC3 = Unchanged.
Error Codes:	ERDIO Disk I/O error. ERSCE Hash file structural error. ERRNF Record not found.

FMSG displays on the current job's message terminal and the system error message associated with the specified system error code. In addition, FMSG will append the file name associated with the specified UFT to the displayed error message.

If messages are suppressed for this terminal, then FMSG does not display anything. If the current job is running in the background and has a predecessor, then FMSG does not display anything.

FMSG sets the current job's error code to the specified error code. The job's error code is passed back to the predecessor job when its execution is restarted at LINK's normal return.

Message

If there is currently no system error message file, the message displayed has the standard format described in the Error Messages manual. If there is a system error message file, the message has the following format:

*program [dsknam A=aaaa H=hhhh] FN=filename message

Fields have variable length as appropriate. Unused fields are not present. Those fields that are present are separated by one blank. Hence, the total message length is minimized.

program is the current job's name. If the current job is the operating system, this name will be "(Cx) SY Rxx.x", where Cx indicates the configuration and Rxx.x indicates the release level. If there is no current job, this field is not present.

[dsknam . . .] appears only on a system error 8 (disk error). Dsknam is the name of the disk on which the error occurred. Aaaa is the logical disk address in hexadecimal on which the error occurred. Hhhh is the hardware status in hexadecimal.

filename is the name of the file associated with the UFT specified in AC2.

The message is displayed on the system message line in the manner described in the System Message Line section. If necessary, the message is truncated. Truncation is not an error and does not cause FMSG to take its error return.

Errors

If there is currently a system error message file and an error occurs in trying to access this file, FMSG returns the appropriate error code after displaying the standard format message for entry parameter AC0. Hence, if there is a system error message file but it does not contain a message for the specified error code, FMSG displays the standard format error message for the error code and then returns error code ERRNF (record not found). Even when FMSG returns an error code, it sets the current job's error code to its entry parameter AC0.

ERROR MESSAGE WITH FILE NAME TO COMPUTER TERMINAL (MSVF)

```
CALL
.WORD MSVF
error return
normal return
```

Entry: AC0 = System error code.

AC1 = Address of File Name Block specifying a computer terminal.

0 ---> display message on the current job's message terminal.

AC2 = Address of UFT.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: AC0-AC3 = Unchanged.

Error Codes:

ERFNF	File not found.
ERCRNO	Terminal not a computer terminal.
ERDIO	Disk I/O error.

MSVF displays on the specified computer terminal an error message for the specified error code and with the name of the file associated with the specified User File Table (UFT).

MSVF sets the current job's error code to the specified error code. The job's error code is passed back to the predecessor job when its execution is restarted at LINK's normal return.

Message

If there is currently no system error message file, the message displayed has the standard format described in the Error Messages manual. If there is a system error message file, the message has the following format:

*program [dsknam A=aaaa H=hhhh] FN=dddddd:ffffff.ff message text

program is the current job's name. If the current job is the operating system, this name will be "(Cx) SY Rxx.x", where Cx indicates the configuration and Rxx.x indicates the release level. This field is always present and always contains 17 bytes or less (including trailing blank).

[dsknam . . .] appears only on a system error 8 (disk error). Dsknam is the name of the disk on which the error occurred. Aaaa is the logical disk address in hexadecimal on which the error occurred. Hhhh is the hardware status in hexadecimal. When this field appears it contains 21 bytes or less (including trailing blank).

dddddd:ffffff.ff is the name of the file associated with the specified UFT. This field contains 20 bytes or less.

The message is displayed on the system message in the manner described in the System Message Line section.

Errors

If the specified File Name Block does not specify an exising file, MSVF returns error code ERFNF. If the specified File Name Block specifies an existing file that is not a computer terminal, MSVF returns error code ERCRNO. If an error occurs in accessing the system error message file, MSVF displays the standard format error message for the specified error code before taking its error return. Even when MSVF takes its error return, it sets the current job's error code to its entry parameter AC0.

DISPLAY MESSAGE (MSG)

```
CALL
.WORD MSG
error return
normal return
```

Entry: AC0 = Address of message.

AC1 = Number of words in message (maximum length D'40 words).

Error Exit: Never used.

Normal Exit: AC0-AC3 = Unchanged.

MSG displays the specified message on the current job's message terminal.

If messages are suppressed for this terminal, then MSG does not display anything. If the current job is running in the background and has a predecessor, then MSG does not display anything.

If the current job is running in the background and does not have a predecessor, then MSG prefixes the name of the current job to the message and truncates the message if necessary.

The message is displayed on the system message line in the manner described in the System Message Line section.

MESSAGE READ AND DISPLAY (MSRD)

```
CALL
.WORD MSRD
error return
normal return
```

Entry: AC0 = Message number. Key to hash record containing message text.

AC1 = Address of message command block.

0 ---> No message command block.

Display the message as specified by the current message parameters for the job message terminal.

AC2 = Address of UFT for message hash file.

0 ---> No message hash file.

-1 --> Use the system error message file.

If AC2 is not 0 or -1, then it must be the address of a UFT.

Error Exit: AC0 = System error code.
AC1 = Number of bytes displayed or put in output buffer.
AC2, AC3 = Unchanged.

Normal Exit: AC1 = Number of bytes in message.
AC0, AC2, AC3 = Unchanged.

Error Codes:

ERLTL	Line too long.
ERHNHF	Not a hash file.
ERCRNO	Not a computer terminal.
ERRNF	Record not found.
ERFNOP	File not open.

Message Command Block Format

Offset	Meaning
0	Display options:

Bits

0 0 --> Do not blink message
1 --> Blink message

1 & 2 Not used if word 2 contains a non-zero value

0 --> Display message on system message line with screen rolling and line blanking as appropriate. See "System Messge Line" description.

1 --> Display the messge in the specified terminal's message area.

2 --> Display message as specified by the message parameters for the terminal used.

3 --> Display message at the screen position specified by bits 4-15 of this word and do not roll the screen or blank any line. The message is truncated at the end of the line.

Offset	Meaning
	Bits
	3 Not used.
	4-8 Line number (range: 0 to 23).
	9-15 Column number (range: 0 to 79).
1	Address of File Name Block specifying a computer terminal.
	0 --> Display message on job message terminal.
2	Address of 41-word output buffer.
	0 --> Display message on terminal. There is no output buffer.
	Non-zero --> Do not display message on terminal. Move message to output buffer.
3	Number of bytes in supplementary text.
	0 --> No supplementary text. In this case the word at offset 4 is not used.
4	Address of supplementary text.

MSRD optionally does any combination of the following:

1. Obtains message text from a message file. See the section "Message File Format."

2. Obtains message text from the system error message file (if any). The system error message file has the same format as a message file.

3. Appends supplementary text to the message text.

4. Displays the message on any specified computer terminal, or moves it to a specified output buffer. If messages are suppressed on this terminal, then MSRD does not display anything. In any event, MSRD returns the number of bytes displayed or moved into the output buffer. This applies to all four options.

If no message command block is specified, the message receives the default message handling for the current job's message terminal. See the System Message Line section for a description of screen handling when the system message line is used.

If input parameter AC2 is zero, the message consists of the supplementary text (if any). If the supplementary text byte count is zero, the message consists of the message text from the message file (if any) or system error file (if any) as specified. If input parameter AC2 is -1, the message text is obtained from the system error message file (if any).

If no 41-word output buffer is specified, the message is displayed on the specified computer terminal. If necessary, the message is truncated at the end of the line. Hence, the maximum number of bytes displayed is D'80 (one full line). If the message command block specifies a screen position with a non-zero column number, the maximum message length is less than D'80. When the message is displayed, the byte count returned does not include a line terminator, because no line terminator is moved to the screen.

If a 41-word output buffer is specified, the message is not displayed, but it is moved into the 41-word output buffer truncated to D'80 bytes if necessary and terminated with a carriage return (X'0D). In this case, the byte count returned includes the line terminator.

If the message is truncated, MSRD displays or moves it to the user's buffer before returning error code ERLTL. If the specified computer terminal can not be found, MSRD returns error code ERFNF. If the specified file is not a computer terminal, MSRD returns error code ERCRNO. If input parameter AC2 is -1 and there is currently no system error message file, MSRD returns error code ERFNOP.

SET OR RELEASE LOWERCASE OPTION (CRTLCI)

```
CALL
.WORD CRTLCI
error return
normal return
```

Entry:

AC0 = 0 Implies translate lowercase characters from the keyboard into uppercase; or use left bytes of the keyboard translation table.

AC0 ≠ 0 Implies do not translate lowercase characters from the keyboard; or use the right bytes of the keyboard translation table.

AC2 = Address of UFT.

Error Return: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Return: Registers unchanged.

Error Codes: ERCRNC CT type call on a non-CT device.

CRTLCI turns the lowercase input option on and off. If there is a keyboard translation table, then no lower to uppercase translation is done. By default keyboard input is translated from lower to uppercase.

Utility program TCUP is used to install or free a keyboard translation table.

SET OR READ TERMINAL STATUS LIGHTS (CRLGTS)

```
CALL
.WORD CRLGTS
error return
normal return
```

Entry Parameters: AC2 = Terminal UFT address from OPEN.

AC0 = -1. Read the current state of the CRT status lights.

AC0 ≠ -1. Update the CRT status lights as indicated by bits 8-15 in AC0.

Exit Parameters: If on input AC0 = -1, current light status is returned in AC0. Bits 0-7 and bit 15 are always 0 on return.

If on input AC0 ≠ -1, no change to registers.

AC0 = Error code if error return.

Error Codes: ERCRCM Terminal in Command mode.

The leftmost of the eight status lights on the J100/J105 is not programmable; it merely indicates that power is on the terminal.

CRLGTS reads or sets the seven programmable lights. If AC0 contains zero when the call is made, the lights are read and their state is returned in bits 8-14 of AC0:

```
LIGHTS:                    • • • • • • •

AC0 :   0 1 2 3 4 5 6 7   8  9 10 11 12 13 14 15

                          Bit = 1   light is ON.
                          Bit = 0   light is OFF.
```

Similarly, setting bits 8-14 on or off in any desired pattern before making the call causes CRLGTS to turn the corresponding lights on or off.

ERASE FREE SCREEN (CEFREE)

```
CALL
.WORD CEFREE
error return
normal return
```

Entry Parameters: AC2 = Address of UFT.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: Registers unchanged.

Error Codes: ERCRMO Mode conflict. The terminal is in roll mode.
              ERCRNC CT type CALL on a non-CT device.

CEFREE erases all of the free part of the specified computer terminal screen. Both protected and unprotected fields are erased. All protect and unprotect characters are blanked. The cursor is left at the Free home position (the upper, left corner of the screen).

ERASE UNPROTECTED FIELDS (CEPROT)

```
CALL
.WORD CEPROT
error return
normal return
```

Entry: AC2 = Address of UFT.

Error Return: AC0 = System error code.
               AC1-AC3 = Unchanged.

Normal Return: Registers unchanged.

Error Codes: ERCRNC CT type call on a non-CT device.

              ERCRPR There is no free home position because the whole screen is protected.

CEPROT erases the unprotected fields in the Free part of the screen if the screen is in Split mode or Free mode. If the screen is in Roll mode CEPROT erases the unprotected fields anywhere on the screen. The cursor is left at the Free home position: the lowest unprotected position. The upper, left corner of the screen is position 0.

ERASE ENTIRE SCREEN (CERALL)

```
CALL
.WORD CERALL
error return
normal return
```

Entry Parameters: AC2 = Terminal UFT address from OPEN.

Exit Parameters: None.
                  AC1-AC3 = Unchanged.

Error Codes: ERCRNC CT type call on a non-CT device.

CERALL erases the entire screen, including the contents of protected fields. The cursor remains at the home position of the line last accessed. It does not return to the bottom of the screen.

ERASE ROLL PART (CEROLL)

```
CALL
.WORD CEROLL
error exit
normal exit
```

Entry Parameters: AC2 = Address of UFT.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.

Normal Exit: Registers unchanged.

Error Codes: ERCRMO Mode conflict. The terminal is in Free Mode.
ERCRNC CT type CALL on non-CT device.

CEROLL erases only the roll part of the specified computer terminal screen. The cursor is left at the Roll home position (the lower left corner of the screen).

KEYBOARD CHARACTER CODES

The next few pages show the character codes generated by the Standard keyboard and by the English-language version of the International keyboard.

The I/O handler for the Standard terminal changes the value of function key <F8> from X'7F to X'06 before the user program actually receives the data. The International keyboard generates X'06 for this key.

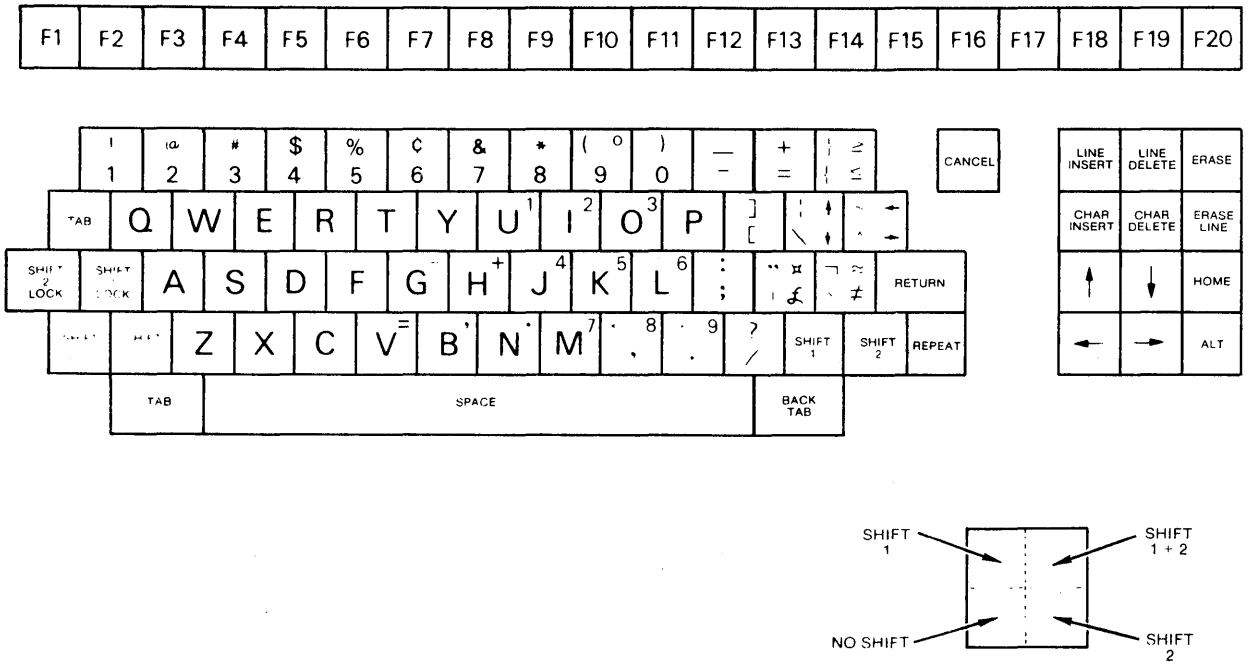
For either keyboard the system automatically converts lowercase letters into uppercase letters before the user program receives the data. If true lowercase input is required, this conversion can be disabled with a CRTLCI system call.

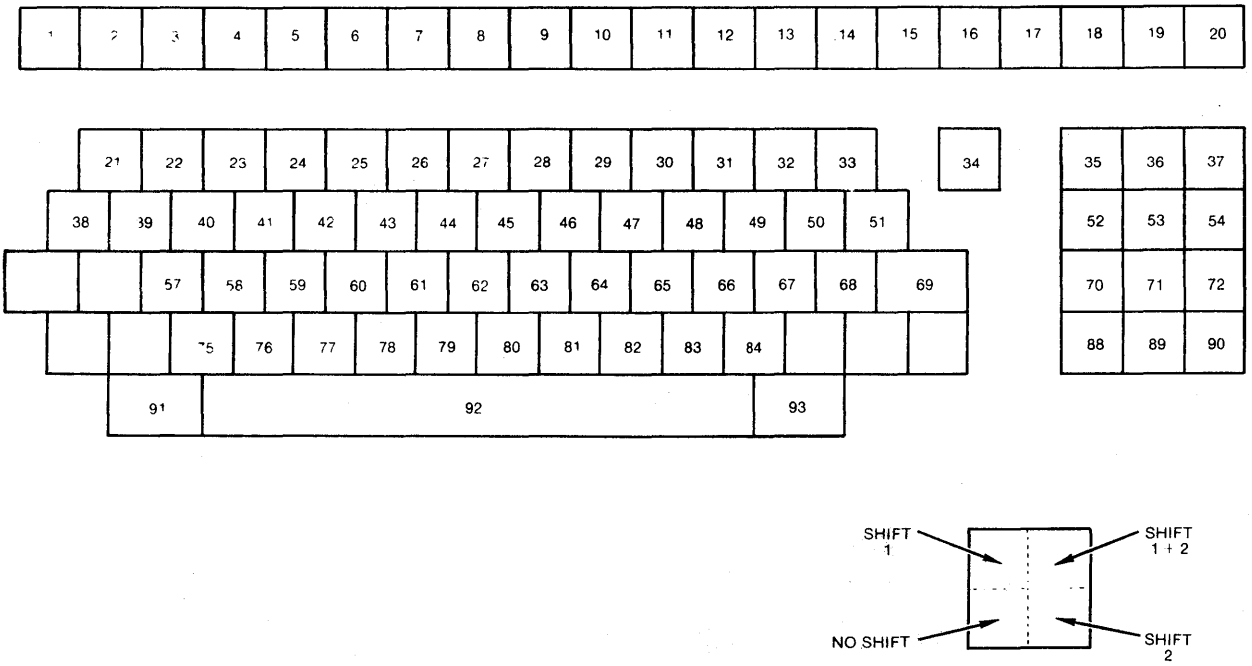
GENERATED CODES - STANDARD KEYBOARD

Key	Hex	Key	Hex	Key	Hex
		@	40	`	60
!	21	A	41	a	61
"	22	B	42	b	62
#	23	C	43	c	63
$	24	D	44	d	64
%	25	E	45	e	65
&	26	F	46	f	66
'	27	G	47	g	67
(	28	H	48	h	68
)	29	I	49	i	69
*	2A	J	4A	j	6A
+	2B	K	4B	k	6B
,	2C	L	4C	l	6C
-	2D	M	4D	m	6D
.	2E	N	4E	n	6E
/	2F	O	4F	o	6F
0	30	P	50	p	70
1	31	Q	51	q	71
2	32	R	52	r	72
3	33	S	53	s	73
4	34	T	54	t	74
5	35	U	55	u	75
6	36	V	56	v	76
7	37	W	57	w	77
8	38	X	58	x	78
9	39	Y	59	y	79
:	3A	Z	5A	z	7A
;	3B	[	5B		
<	3C	\	5C		
=	3D	]	5D		
>	3E	^	5E		
?	3F	_	5F		

<ERASE> or <RULER/PRINT>	03	<F1>	1C	
<ERASE LINE>	04	<F2>	1D	
<HOME>	07	<F3>	1E	
<CANCEL>	08	<F4>	1F	
<TAB>	09	<F5>	0B	
<RETURN>	0A	<F6>	1B	
<BACK TAB>	0C	<F7>	05	
<CHAR DELETE>	17	<F8>	7F	(becomes 06)
<CHAR INSERT>	18	<F9>	01	
Space Bar	20	<F10>	10	
<|> (vertical) or <CENTER>	7C			
<DOWN ARROW>	00			
<UP ARROW>	02			
<RIGHT ARROW>	1A			
<LEFT ARROW>	19			

LAYOUT - MODEL 4800 ENGLISH KEYBOARD





GENERATED CODES - MODEL 4800 ENGLISH KEYBOARD (PART 1)

	SHIFT 1+2		SHIFT 2		SHIFT 1		NO SHIFT	
KEY NO.	KEY SYM	HEX CODE	KEY SYM	HEX CODE	KEY SYM	HEX CODE	KEY SYM	HEX CODE
1	F1	1C	F1	1C	F1	1C	F1	1C
2	F2	1D	F2	1D	F2	1D	F2	1D
3	F3	1E	F3	1E	F3	1E	F3	1E
4	F4	1F	F4	1F	F4	1F	F4	1F
5	F5	0B	F5	0B	F5	0B	F5	0B
6	F6	1B	F6	1B	F6	1B	F6	1B
7	F7	05	F7	05	F7	05	F7	05
8	F8	06	F8	06	F8	06	F8	06
9	F9	01	F9	01	F9	01	F9	01
10	F10	10	F10	10	F10	10	F10	10
11	F11	F1	F11	F1	F11	F1	F11	F1
12	F12	F2	F12	F2	F12	F2	F12	F2
13	F13	F3	F13	F3	F13	F3	F13	F3
14	F14	F4	F14	F4	F14	F4	F14	F4
15	F15	F5	F15	F5	F15	F5	F15	F5
16	F16	F6	F16	F6	F16	F6	F16	F6
17	F17	F7	F17	F7	F17	F7	F17	F7
18	F18	F8	F18	F8	F18	F8	F18	F8
19	F19	F9	F19	F9	F19	F9	F19	F9
20	F20	FA	F20	FA	F20	FA	F20	FA
21		21		21	!	21	1	31
22		40		40	@	40	2	32
23		23		23	#	23	3	33
24		24		24	$	24	4	34
25		25		25	%	25	5	35
26		83		83	¢	83	6	36
27		26		26	&	26	7	37
28		2A		2A	*	2A	8	38
29	ø	30		28	(	28	9	39
30		29		29	)	29	ø	30
31		5F		5F	_	5F	–	2D
32		2B		2B	+	2B	=	3D
33	≥	81	≤	80	}	7D	{	7B
34	CANCEL	08	CANCEL	08	CANCEL	08	CANCEL	08
35	LINE INS	16	LINE INS	16	LINE INS	16	LINE INS	16
36	LINE DEL	15	LINE DEL	15	LINE DEL	15	LINE DEL	15

GENERATED CODES - MODEL 4800 ENGLISH KEYBOARD (PART 2)

	SHIFT 1+2		SHIFT 2		SHIFT 1		NO SHIFT	
KEY NO.	KEY SYM	HEX CODE	KEY SYM	HEX CODE	KEY SYM	HEX CODE	KEY SYM	HEX CODE
37	ERASE	03	ERASE	03	ERASE	03	ERASE	03
38	TAB	09	TAB	09	TAB	09	TAB	09
39		51		51	Q	51	q	71
40		57		57	W	57	w	77
41		45		45	E	45	e	65
42		52		52	R	52	r	72
43		54		54	T	54	t	74
44		59		59	Y	59	y	79
45	1	31		31	U	55	u	75
46	2	32		32	I	49	i	69
47	3	33		33	O	4F	o	6F
48		50		50	P	50	p	70
49		5D		5D	]	5D	[	5B
50	↑	E5	↓	E7	¦	7C	\	5C
51	←	E8	→	E6	~	7E	^	5E
52	CHAR INS	18	CHAR INS	18	CHAR INS	18	CHAR INS	18
53	CHAR DEL	17	CHAR DEL	17	CHAR DEL	17	CHAR DEL	17
54	ERASE LINE	04	ERASE LINE	04	ERASE LINE	04	ERASE LINE	04
55	SHIFT 2		SHIFT 2		SHIFT 2		SHIFT 2	
56	SHIFT 1		SHIFT 1		SHIFT 1		SHIFT 1	
57		41		41	A	41	a	61
58		53		53	S	53	s	73
59		44		44	D	44	d	64
60		46		46	F	46	f	66
61	–	2D		2D	G	47	g	67
62	+	2B		2B	H	48	h	68
63	4	34		34	J	4A	j	6A
64	5	35		35	K	4B	k	6B
65	6	36		36	L	4C	l	6C
66		3A		3A	:	3A	;	3B
67	¤	E4	£	85	"	22	'	27
68	≈	E3	≠	82	¬	7F	\	60
69	RETURN	0A	RETURN	0A	RETURN	0A	RETURN	0A
70	CURS UP	02	CURS UP	02	CURS UP	02	CURS UP	02

GENERATED CODES - MODEL 4800 ENGLISH KEYBOARD (PART 3)

	SHIFT 1+2		SHIFT 2		SHIFT 2		NO SHIFT	
KEY NO.	KEY SYM	HEX CODE	KEY SYM	HEX CODE	KEY SYM	HEX CODE	KEY SYM	HEX CODE
71	CURS DWN	00	CURS DWN	00	CURS DWN	00	CURS DWN	00
72	HOME	07	HOME	07	HOME	07	HOME	07
73	SHIFT 2		SHIFT 2		SHIFT 2		SHIFT 2	
74	SHIFT 1		SHIFT 1		SHIFT 1		SHIFT 1	
75		5A		5A	Z	5A	z	7A
76		58		58	X	58	x	78
77		43		43	C	43	c	63
78	=	3D		3D	V	56	v	76
79	,	2C		2C	B	42	b	62
80	.	2E		2E	N	4E	n	6E
81	7	37		37	M	4D	m	6D
82	8	38		38	<	3C	,	2C
83	9	39		39	>	3E	.	2E
84		3F		3F	?	3F	/	2F
85	SHIFT 1		SHIFT 1		SHIFT 1		SHIFT 1	
86	SHIFT 2		SHIFT 2		SHIFT 2		SHIFT 2	
87	REPEAT		REPEAT		REPEAT		REPEAT	
88	CURS LT	19	CURS LT	19	CURS LT	19	CURS LT	19
89	CURS RT	1A	CURS RT	1A	CURS RT	1A	CURS RT	1A
90	ALT MODE	0F	ALT MODE	0F	ALT MODE	0F	ALT MODE	0F
91	TAB	09	TAB	09	TAB	09	TAB	09
92	SPACE	20	SPACE	20	SPACE	20	SPACE	20
93	BACK TAB	0C	BACK TAB	0C	BACK TAB	0C	BACK TAB	0C

INPUT ECHOING - COMMAND AND ROLL MODES

If the terminal is in either Command Mode or Roll Mode, the following response occurs for the various keys. No echoing occurs in the Roll Mode unless the terminal is open and an input operation is pending. All echoing appears on the bottom line of the screen.

<F1> through <F10>	No response.
<F11> through <F20>	International keyboard only. No response.
ASCII Characters	Displayed at the current cursor location. Moves the cursor one position to the right. Wraps around the bottom line as needed.
<ERASE>	Erases the entire screen. The cursor then appears at the start of the bottom line.
<CHAR INSERT>	Character Insert. Causes all characters from the current cursor position to be shifted right one position. Any character which was on the righthand edge of the screen is lost.
<CHAR DELETE>	Character Delete. Causes all characters from the current cursor position to be shifted left one position. A blank is supplied for the last position on the line.
<ERASE LINE>	Erases all characters from the current cursor position to the end of the line.
<UP ARROW>	No effect.
<DOWN ARROW>	Rolls the screen down one line.
<HOME>	Returns the cursor to the start of the bottom line.
<LEFT ARROW>	Moves the cursor one position to the left. Wraps around the bottom line as needed.
<RIGHT ARROW>	Moves the cursor one position to the right. Wraps around the bottom line as needed.
<CANCEL>	No effect in Command Mode. In Roll mode, it aborts the job attached to the terminal, unless an ABTC system call is in effect.
<RETURN>	Causes information on the bottom line to be transferred to the system if in Command Mode. In Roll Mode, it completes the application program's input operation.

<TAB> Moves the cursor to the next tab stop. (Tab stops are located at every eighth column.) After the last tab stop, the cursor wraps around to the home position.

<BACK TAB> Moves the cursor to the preceding tab stop. No effect if the cursor is at the home position.

<ALT> Universal keyboard only. No effect.

INPUT ECHOING - FREE SCREEN MODE

If the terminal is in the Free Screen Mode, the following response occurs for the various keys - assuming that a program is doing a read (RCRT) operation. Echoing occurs anywhere on the screen.

<F1> through <F10> Terminate the read and return the function key code to the user program:

<F1>	X'1C	<F6>	X'1B
<F2>	X'1D	<F7>	X'05
<F3>	X'1E	<F8>	X'06 *
<F4>	X'1F	<F9>	X'01
<F5>	X'0B	<F10>	X'10

*The Standard keyboard generates X'7F for <F8>, but the system converts this to X'06 - the same code generated by the International keyboard.

<F11> through <F20> International keyboard only. Terminate the read and return the function key code to the user program:

<F11>	X'F1	<F16>	X'F6
<F12>	X'F2	<F17>	X'F7
<F13>	X'F3	<F18>	X'F8
<F14>	X'F4	<F19>	X'F9
<F15>	X'F5	<F20>	X'FA

ASCII Characters Displayed at the current cursor location. The cursor then moves right one position.

<ERASE> Erases all unprotected data from the free screen. The cursor does not move.

<CHAR INSERT> Character Insert. Causes all characters from the current cursor position to be shifted right one position. Characters are not shifted into protected fields. The shift stops at the end of a line or at the first protected character encountered. The last character, if shifted into the end of the line or into a protected position, is lost.

<CHAR DELETE>	Causes all characters from the current cursor position to be shifted to the left one character position. The shift stops at the end of a line or if a protected field is encountered. A blank is used as the fill character.
<ERASE LINE>	Erases all characters from the current cursor position to the first protected character or to the end of the screen, whichever occurs first.
<UP ARROW>	Moves the cursor up one line. If that position is protected, it moves left until an unprotected position is found. The cursor wraps around from the top line to the bottom line as needed.
<DOWN ARROW>	Moves the cursor down one line. If that position is protected, it moves right until an unprotected location is found. The cursor wraps around from the bottom line to the top line as needed.
<HOME>	Moves the cursor to the first unprotected position on the screen.
<LEFT ARROW>	Moves the cursor one position to the left. If that position is protected, it moves left until an unprotected position is found. The cursor wraps around to the next line as needed.
<RIGHT ARROW>	Moves the cursor one position to the right. If that position is protected, it moves right until an unprotected position is found. The cursor wraps around the line as needed.
<CANCEL>	Aborts the job attached to the terminal, unless an ABTC system call is currently in effect.
<RETURN>	The cursor is moved to the beginning of the next line. If that position is protected, it is moved right until an unprotected position is found. Wraparound occurs at the end of the screen.
<TAB>	If there are no unprotected fields, the cursor does not move. Otherwise, the cursor is moved right-wards to the start of the next unprotected field; wraparound occurs at the end of the free part of the screen.
<BACK TAB>	If there are no protected fields, the cursor does not move. Otherwise, it is moved to the left until it is at the beginning of an unprotected field. If it was already at the start of an unprotected field, it moves to the start of the preceding unprotected field. Wraparound occurs from the beginning of the free part to its end.

<ALT>	Universal keyboard only. Terminates the read and returns X'0F to the user program.

INPUT ECHOING - FIELD ENTRY

<F1> - <F20>	Completes the current field, inhibit input, and returns the function key code to the user program.
ASCII characters	Displayed at the current cursor location. Moves the cursor one position to the right. If this position is protected, the cursor is positioned at the start of the next unprotected field to the right.
<ERASE>	Erases the current unprotected field and positions the cursor to the start of this field.
<UP ARROW>	Ignored.
<DOWN ARROW>	Ignored.
<HOME>	Positions the cursor to the start of the field.
<LEFT ARROW>	Moves the cursor one position to the left, unless the cursor is at the start of the field in which case <LEFT ARROW> is ignored.
<RIGHT ARROW>	Moves the cursor one position to the right. If this position is protected, the cursor is positioned at the start of the next unprotected field to the right.
<RETURN>	Ignored.
<TAB>	The cursor is positioned at the start of the next unprotected field to the right.
<BACK TAB>	Cursor is positioned at the start of the current field.

INPUT ECHOING - SINGLE-FIELD READ

<F1> - <F20>	Completes the current field, inhibits input, and returns the function key code to the user program.
ASCII characters	Can not complete the current field, can not move the cursor out of the current field. At the end of the current field, overstriking occurs.
<ERASE>	Erases the current unprotected field and positions the cursor to the start of this field.
<UP ARROW>	Ignored.

<DOWN ARROW>	Ignored.
<HOME>	Positions the cursor to the start of the field.
<LEFT ARROW>	Moves the cursor one position to the left, unless the cursor is at the start of the field in which case <LEFT ARROW> is ignored.
<RIGHT ARROW>	Moves the cursor one position to the right, unless this position is protected. Can not move the cursor out of the current field. Can not complete the current field.
<RETURN>	Ignored.
<TAB>	Completes the current field and inhibits input.
<BACK TAB>	Cursor is positioned at the start of the current field.

SCREEN CONTROLS

Computer terminal screens have display features which are controlled by special characters. These control characters occupy screen positions but, except for certain underscore controls, they display as blanks.

The following table lists the display features and the hexadecimal value of the associated control characters. For each of the features, a "5" in the N/A column indicates that it is not available on the J100; a "1S" indicates that the feature is not available on the J100/J105/J50 with Standard keyboard controllers.

Feature	Hex Code	N/A
Half intensity starts	11	5
Half intensity ends	12	5
Blinking starts	13	
Blinking ends	14	
Inversion starts	15	1S
Inversion ends	16	1S
Blanking starts	1A	1S
Blanking ends	1B	1S
Protected area starts	1E	
Protected area ends	1F	
Underscoring starts	17	1S
Underscoring ends	18	1S
Underscoring starts (with marker on J100)	10	1S
Underscoring ends (with marker on J100)	19	1S

A control character affects all the screen positions following it, until it is either reversed by another code or removed from the screen by further output. A screen area may be affected by several unrelated codes.

Inversion causes each character to be displayed as dark dots against a light background. The X'16 code itself will appear as an inverted blank; the X'17 will be a normal blank. Blanking makes an area invisible without disturbing its actual contents. Protected areas affect the processing of keyboard editing operations, as described earlier.

On a J500, underscoring affects all screen positions between X'17, which appears as an underscored blank, and X'18, which appears as a normal blank.

On a J100 with an International keyboard, both X'17 and X'18 appear as normal blanks. Underscoring affects only characters in the range X'21 through X'E6; an ASCII blank (X'20) is not underscored.

X'10 and X'19 appear as right-triangular markers (sloping left and right respectively) on a J100 with a International keyboard. In all other respects, they are equivalent to X'17 and X'18.

Chapter 10

COMMUNICATIONS AND ASYNCHRONOUS CHARACTER I/O

Standard System II software includes drivers for printers, for auto-dial units, and for both asynchronous and synchronous communications devices. Jacquard also offers specialized communications drivers and emulators for line protocols such as IBM 2780/3780, IBM 3270, and UNIVAC 1004; these software products must be ordered separately.

To reduce the amount of memory required by the system when printers or communications devices are not in use, the drivers and emulators reside on disk. System II loads the appropriate driver in response to an "OPEN" request for the relevant device.

System II provides a common device-independent interface to all supported devices. To conform to this standard, the printer and communications device drivers perform the following functions:

- Interrupt level I/O processing.
- Buffering data input from the device.
- Buffering data to be output to the device.
- Error checking on input, and error recovery where possible.
- Character code translation, as described in Chapter 15.
- Specialized protocol handling where applicable.

This section describes the asynchronous character device drivers. Other communications drivers are described in the publications cited in the following summary.

ASYNCHRONOUS DRIVERS

Driver	Device	Description
CDLDRV.SB	SILA (Synchronous-Isochronous Adapter)	J100 only. Asynchronous input and output using SILA.
CDODRV.SB	Line Printer Controller	Output only. Included in standard software.
CDXDRV.SB	Asynchronous Multiplexer, Single Line	Input and output for remote terminals. Included in standard software.
MUXDRV.SB	Asynchronous Multiplexer, Eight Lines	J100 only. Input and output for multiple remote terminals or printers. Included in standard software.
DPRDRV.SB	Character Printer	Output only. Included in standard software.
DILDRV.SB	Auto-Dialer	J500 only. Included in standard software.

SYNCHRONOUS DRIVERS

Driver	Device	Description
SILDRV.SB SIL5DR.SB	SILA	General synchronous drivers for SILA and user-written communications protocols. Included in standard software. Write-up on request.
UJEDRV.SB	SILA	Driver used alone or with URJE emulator (UNIVAC 1004 RJE). Driver and emulator in "URJE" package; manual V1-066.
CSEA00.SB	SILA	BSC (Binary Synchronous Communications) drivers, multi-point and point-to-point, for ASCII or EBCDIC controls. Used alone or with BRJE utility. Drivers and BRJE in "CSA" package; manual V1-066.

ASYNCHRONOUS COMMUNICATIONS DRIVERS

The three asynchronous communications drivers are CDXDRV.SB for the single-line asynchronous controller, and MUXDRV.SB for the 8-line asynchronous multiplexer and CDLDRV for SILA. The asynchronous controllers support communications with devices such as remote terminals and printers, as well as communication between computers. Support is provided only for devices included in the current AM Jacquard Systems sales catalog.

Asynchronous drivers process standard system calls: OPEN, OPNW, OPNR, CLOS, WRL, WRS, RDL, and RDS (see Chapter 7). Any code translation occurs automatically just after input, or just before output, as described in Chapter 15.

In addition to the usual RDS/RDL errors, these codes are possible:

ERCMBO Buffer overflow
ERCMPE Parity error
ERCMFE Framing error
ERCMNR Data set not ready

When such an error occurs, the last character delivered to the user program (and included in the AC1 byte count) is the one which caused the error. To obtain the rest of the buffer, another read is needed.

Both hardware and software options control the manner in which characters are handled. Controller hardware options include:

- Character size in bits
- Character parity generation and checking
- Number of start and stop bits
- Baud rate

In addition, modems, remote terminals, and printers may have switches for several of the hardware options above, as well as half/full duplex, generation of XON/XOFF, and spacing/form feed. Controller hardware options, terminal or printer options, and device characteristics chosen during system generation must be considered together when installing hardware and software for these devices.

Hardware and software impose some restrictions on the options which may be selected in the hardware and during system generation for asynchronous character I/O.

SOFTWARE RESTRICTIONS

Standard software does not support screen operations in Free Mode or Split Mode for remote CRTs on asynchronous lines, as opposed to J105 satellites, which use main memory refresh.

This restriction excludes not only user-written programs in BASIC or assembly language, it also excludes Type-Rite, the Account-Rite packages (General Ledger, Payroll, etc.), any application created with Data-Rite, and standard utilities with full screen displays (such as FILES, JOBS, PARTS, DSKOPY, EDIT, and interactive SORT).

BAUD RATE

The practical upper limit is 4800 baud for a J100 and 9600 baud for a J500; however, baud rate really depends upon the application and the total hardware configuration. When an asynchronous line is used to transfer data to a device without long pauses between records or groups of characters, the baud rate must suit the receiving device. If data is being copied to a floppy disk the maximum is 300 baud. For a cartridge disk the maximum is 1200 baud.

The maximum baud rate for output to a remote character printer is affected by the printer's response to commands such as carriage return and form feed. The asynchronous drivers have no means of detecting that the remote printer is busy with one of these time consuming commands and will continue to transmit characters. The printer may then behave peculiarly or lose characters.

The Diablo HyTerm 1610 operates correctly at 1200 baud. The Texas Instruments 810 prints correctly at 300 baud; in some cases it may run correctly at 1200.

DUPLEX MODES

All asynchronous drivers and associated hardware operate in full-duplex mode. The J100 CDLDRV driver and all J500 asynch drivers will also operate in half duplex mode. In general, all printers and remote CRTs should operate full duplex.

X-OFF AND X-ON

In some full duplex circuits, data flow is regulated with two characters called "X-OFF" and "X-ON" on a teletypewriter keyboard.

Sending X-OFF (or "Control/S" - DC3 in ASCII, X'13) requests "quiet" - the other end of the line should halt its transmission.

Sending X-ON (or "Control/Q" - DC1 in ASCII, X'11) indicates "restart" - the other end of the line may resume its transmission.

In System II, this convention may be enabled for a full-duplex asynch device by setting the XO characteristic, either at system generation time or through the ACUP utility. The following guidelines should be observed:

- Only ASCII can be used, and neither X-OFF nor X-ON can appear as data characters.
- The minimum acceptable buffer size is 32 characters.
- Several X-OFF or X-ON characters can be sent consecutively for a single "quiet" or "restart" request.

- If one end of the line is echoing all the data received, it is mandatory for the other end to read back at least as many characters as it writes. Furthermore, the writes must not proceed faster than the reads; in other words, echoing implies that characters must not be sent faster than they can be received.

AUTO-DIALER DRIVER

DILDRV.SB is a J500 device driver for initiating a telephone call on either a synchronous or an asynchronous communications line.

The required hardware connections are:

1. J500 auto-dialer port to Bell 801 (or equivalent) auto-dial unit. This is an RS-366 connection.

2. J500 communications port to a dialer-compatible modem. The System II device on this line must be defined with the DC option; that is, disconnected when closed.

3. Phone line (through a DAA) to both the modem and the dialer.

Programming a single telephone call involves the following steps:

1. Open the dialer and the communications device; it does not matter which is opened first.

2. Use a WRL or WRS call to output the phone number to the dialer. The caller will be suspended until one of these events occurs:

   - The call is completed successfully.

   - The dialer abandons the call. Typical reasons are dialer timeout, no dial tone, busy signal, or out-of-service signal.

   - No response from the dialer for two minutes.

3. For a normal return from the WRL/WRS, close the dialer. The communications device will then be connected to the phone line.

4. For an error return from the WRL/WRS, close the dialer. After a suitable real-time wait (no less than 15 seconds), re-open the dialer and go to step 2.

Each digit of the the phone number is normally an ASCII numeral, from "0" (X'30) to "9" (X'39), or a punctuation mark from X'3A to X'3F, which some dialers will recognize as a pause for another dial tone. With the exceptions noted below, other values from X'10 to X'FF are also acceptable. In any case, the low-order four bits are taken as the digit to be passed to the dialer.

Values X'20 (space), X'2D (hyphen), and X'09 (horizontal tab) are ignored; nothing is passed to the dialer.

The string must end with a line terminator, typically X'0D, although any value less than X'10, except X'09, is also a valid terminator.

The error codes generated by DILDRV.SB are ERICL, ERISZ, ERABT, and ERLTL, plus the unique auto-dialer codes ERDLNP, ERDLAC, and ERDLTO.

PRINTER DRIVERS

As defined at SYSGEN time, local printers and their controllers use one of the printer drivers, CDODRV.SB (Data Products, Centronics, Printronix, Texas Instruments) or DPRDRV.SB (Qume, Xerox, NEC, or Diablo other than HyTerm). Remote or asynchronous printers require an asynch driver, as described previously. In any case, support is provided only for printers included in the AM Jacquard sales catalog.

All printer drivers process standard system calls: OPNW, CLOS, WRL, and WRS (see Chapter 7). In general, when default characteristics have been chosen during system generation, a line terminator, a character less than X'10 except for Horizontal Tab (X'09) and Form Feed (X'0C), is converted to a Carriage Return (X'0D) followed by a Line Feed (X'0A). The drivers respond to X'0C with a page eject and to X'09 by generating blanks for tab expansion. Data code translation through a "CT" file, as described in Chapter 15, is also supported.

Driver behavior is influenced by SYSGEN device characteristics. For instance, by selecting transparent mode (TM) or extended character set (ES), the user can output any bit pattern without interpretation by the driver. Some line printers have "built-in" features, such as automatic form feed every 10 inches. To eliminate such features, the hardware must be modified, and the characteristic must be omitted from those selected during system generation.

For a printer using DPRDRV.SB (or for a Diablo HyTerm using an asynch driver), two special control sequences are recognized: X'1B1F' (Set Horizontal Pitch) and X'1B1E' (Set Vertical Pitch). The next output character after the sequence specifies the pitch.

Horizontal pitch, in characters per inch:

X'82	120 cpi	X'8B	12 cpi	X'99	5 cpi
87	20	8D	10	9F	4
88	17	90	8		
89	15	95	6		

Vertical pitch, in lines per inch:

X'82	48 lpi	X'89	6 lpi	X'91	3 lpi
87	8	8B	5		
88	7	8D	4		

For Diablo HyType and Qume Sprint 3 printers, 96-character wheels include a position corresponding to X'20. Because this value is defined as a space by ASCII and by System II, the drivers accept X'10 instead to print this

position. Two more control codes are defined for these printers: X'17 (Start Underscore Mode) and X'18 (End Underscore Mode). Only non-blank positions are underscored, and the control codes themselves will appear in the output as spaces.

PRINTER WHEEL TABLE USAGE

Printer wheel tables are provided for translation of characters displayed on the CRT screen, causing the print-wheel/thimble to strike at the correct character position. The tables also give information on printer hammer intensity and ribbon advance. For Release 9.0, they are contained on diskette TBL190. Those tables which are needed should be copied to the primary disk.

(THIS PAGE INTENTIONALLY BLANK)

Chapter 11

MAGNETIC TAPE FILE MANAGEMENT

INTRODUCTION

The magnetic tape facilities of the System II operating system have been designed to support magnetic tape devices with an absolute minimum of effort and yet allow enough flexibility to permit intimate control of the device if desired.

Two independent types of support for magnetic tape files are available in the operating system. The first type provides standard system sequential file I/O on 9-track tape devices. Chapter 7 details the standard system sequential file I/O philosophy; however, the support for tape files has been expanded sufficiently to justify that a separate section be included to explain these facilities. The second type of support provides device level control over 9-track tape devices by means of driver calls.

All of the facilities to be described in this section are incorporated in a binary file named MT9DRV.SB which is disk resident. This file must be available on the system primary disk when a 9-track magnetic tape device is opened. It is loaded into memory at "open-time" and will remain resident until all 9-track tape devices are closed. The MT9DRV.SB file is sharable (/ES) and only one copy is required for concurrent support of all 9-track tape devices which have been configured in the system (via system generation).

A system user (application program) must select which of the two types of tape support is more suitable since they cannot be arbitrarily intermixed. As a rule of thumb, sequential file I/O should always be used unless more flexibility is needed to deal with unusual tape formats. The support type is dynamically selected via the open call:

Open Call	Support Type
OPEN	Driver level calls
OPNR	Sequential file I/O (input)
OPNW	Sequential file I/O (output)

SEQUENTIAL FILE I/O OPEN

An open call must be made for a tape file before activity of any kind may occur on that file. The open call for a magnetic tape device is in the standard form described in Chapter 7 with certain optional extensions. The following information is defined by the open call:

- Sequential file I/O support
- Input or output file
- Share/no share output file
- Tape mode
    1. Undefined length records
    2. Variable length records
    3. Fixed length records
- Physical block size
- Logical record size
- Code translation
- Initial volume position

The specific form for the open call is:

```
Input:  CALL                    Output:  CALL
        .WORD OPNR                       .WORD OPNW
        error return                     error return
        normal return                    normal return
```

The open call, on normal return, provides a User File Table (UFT) address in AC2. This UFT must be supplied to every subsequent file service call since it identifies the file to the system. A UFT should be used only by a single task; multiple tasks should perform multiple opens so that each has its own UFT.

The open calls for both input and output files require the following entry parameters:

AC1 = Flags.

AC0 = File name block (FNB) address.

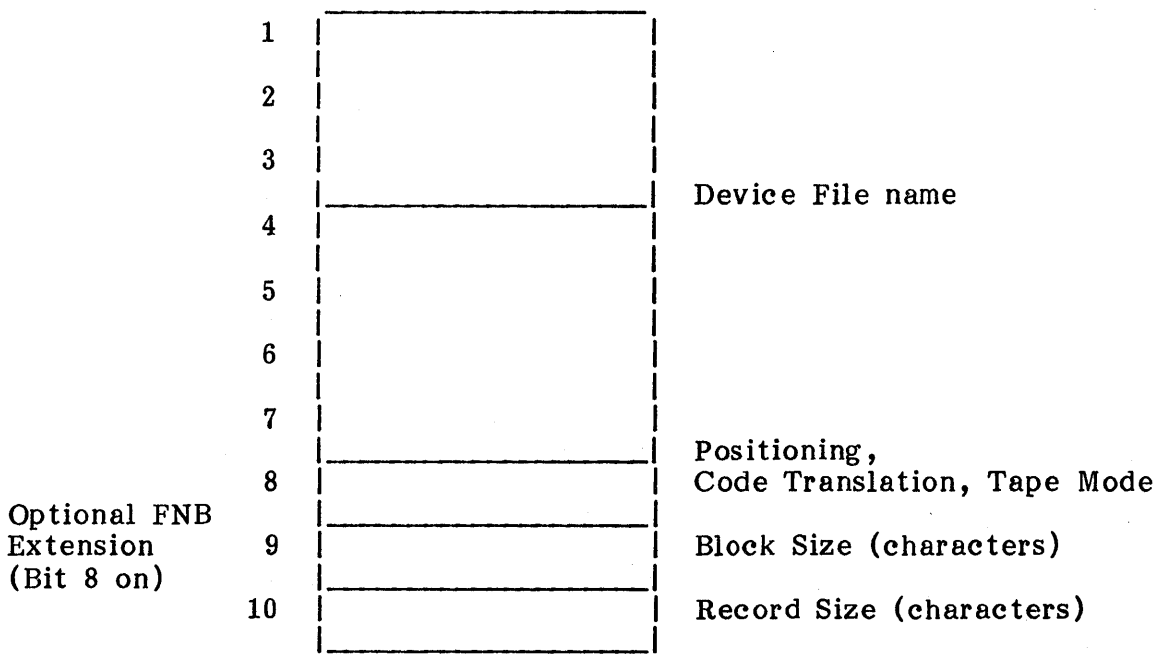
The flags provided in AC1 specify file share information and whether an extended FNB is being used in the call.

Flag Bit	Description
14	Write share output tape file.
8	Extended FNB specified in AC0.

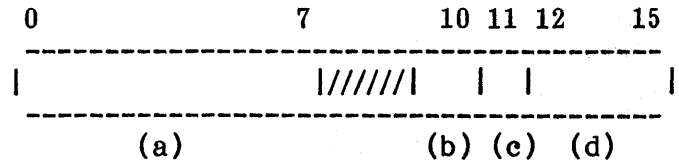
Since 9-track magnetic tape devices are configured into the operating system as write sharable devices, an output file on magnetic tape can be opened concurrently by multiple jobs. Each such job may then act as an independent write process and the collective output will be pooled into a single sequential file on tape. Records will be added to a shared file on a "first come" basis. Interleaving can occur only at a record level; each record will be contiguous within the file.

Bit 14 of the flags (AC1) must be set only for an output file (OPNW) and then only to indicate the user is willing to share the output tape file with other write processes that may have already opened the file or are yet to request file open. If this bit is not set, the open call will succeed only for the first open; subsequent open requests will be refused since the file has been opened non-sharable.

Bit 8 of the flag word, when set, indicates that three extra words appear at the end of the standard 7-word FNB whose address is specified in AC0. The first seven words must contain the device file name in standard form (in words 4, 5, and 6). An extended FNB must appear as follows:



The eighth word of the FNB (if it appears) must have this format:



(a) Bits 0-7 — 0 - No skip (default).

n - Skip n EOF marks where n is a signed integer. Negative implies motion toward the load point.

(b) Bit 10 — 0 - No rewind (default).
1- Rewind before any EOF skips.

(c) Bit 11 — 0 - No translate (default).
1 - Translate.

(d) Bits 12-15 — Mode: 0 - Variable (default).
1 - Undefined
2 - Variable
3 - Fixed

An initial tape volume position may be established for an unshared file by use of the rewind and skip fields in the eighth word. Volume positioning requests on a write share open will result in an error return. The rewind bit (10) is inspected first and, if set, causes the volume to be rewound to load point. The skip field (bits 0-7) is a signed integer (-128 to +127) which, if non-zero, causes the specified number of EOF marks to be skipped. A negative count causes motion towards the load point.

The code translation option is selected by setting bit 11 in word 8 of an extended FNB. If code translation is specified, a file with this name

$device.CT

must be available, or the OPEN will fail with an error code 34. A standard file named EBCDIC.CT is supplied. If translation is required, a copy of that file (with an appropriate name, such as MTOO.CT) could be employed to effect translation from ASCII to EBCDIC during output, and translation from EBCDIC to ASCII during input.

For further details on "CT" files, refer to Chapter 15.

The mode field (bits 12-15) defines the basic physical structure of the tape file. In brief, these modes have the following characteristics:

- Undefined length records (U-mode)

  No information is implied at open time about the tape file block size. For an output file, each record output (each WRS, WRL call) will be written as a separate physical block on tape. Any translation and the device data transfer occur from the user's data area. Similarly, for input files, each record input (each RDS, RDL call) will cause the next physical block on tape to be read directly into the user's data area. Any translation also occurs in the user's data area. This mode requires the least overhead since there is no blocking/deblocking in a system buffer area.

- Variable length records (V-mode)

  No information is implied at open time about the tape file logical record size; however, for output files, records will be blocked to the defined block size before being written to tape. As each record is output (WRS, WRL call), it is added to the current block. Any data which does not fit in the current block will be continued in the next block. For input files, records will be deblocked based on delimiters (terminators) on tape or based on character counts. V-mode is the default if no mode is supplied.

- Fixed length records (F-mode)

  Both the physical block size and the logical record size are defined, where the block size is an integer multiple of the record size. This integer is the blocking factor. For output files, each record is either truncated or padded with blanks to agree with the record size. These records are blocked to the specified block size in a system buffer area and ultimately written to tape. For input files, records are deblocked according to the declared record size. Each input request (RDS, RDL) causes an entire record to be obtained.

In an extended FNB, the ninth word, when non-zero, specifies a physical block size in characters; this block size must be positive. This field is ignored for U-mode tape operation. When the field is zero, a default value of 800 characters will be supplied for V- or F-mode tape files.

The tenth word when non-zero specifies a logical record size in characters. This record size must be positive, and it must divide the block size evenly. This field is required only for F-mode tape operation and ignored otherwise. When the field is zero, a default value of 80 characters will be supplied for an F-mode tape file.

An open call made with no extended FNB will result in a tape file with the following default characteristics:

- No volume positioning
- No translation
- V-mode
- 800 character block size

Note: Block size can be altered by using the MCUP utility. See the System II Utilities Manual for additional information.

A variety of error conditions may occur during open processing. Certain general system errors are itemized in Chapter 7. The following itemizes several additional error codes (and their causes) that may result for sequential file I/O open requests on a 9-track magnetic tape device.

ERTIO - TAPE I/O ERROR

This code is generally reserved for errors involving the device itself. Possible causes are:

Tape Device Status	Results if the device is offline, not ready, or some abnormal status occurs (see UFT description in this section).
Rewind Error	Results if an abnormal status occurs during an attempt to rewind the tape volume (FNB word 8, bit 10).
Skip EOF Error	Results if the requested tape position cannot be established due to load point, EOT marker, or an abnormal status (FNB word 8, bits 0-7).
File Protect	For an output file only; results if no write ring is on the tape volume, causing a file protect status indication.

ERTPE - TAPE PARAMETER ERROR

This code is generally reserved for errors involving parameters and system level rules. Possible causes are:

Block Size	Results if the block size in characters is not positive (V- or F-mode only).
Record Size	Results if the record size in characters is not positive or does not divide the block size evenly (F-mode only).
Share Conflicts	Results on write share opens when volume positioning is requested or when the following parameters do not agree with those declared by previous opens: Mode Block Size (V, F only) Record Size (F only) Translation

ERNPA - NO PARTITION AVAILABLE

This is a standard system error that may occur on V- or F-mode opens only, or any open specifying translation. It indicates that insufficient contiguous main memory was available to allocate tape I/O buffer space, or translation table space. The amount requested for an I/O buffer is the declared block size. For a shared device, this may occur only on the first open.

PHYSICAL TAPE STRUCTURE AND DATA TRANSFER

Magnetic tape often serves as a medium for the exchange of information between different types of computer systems. As a result, it is often important to know the exact physical structure of files on tape volumes. A discussion follows which details the structures that the System II tape facilities can handle.

A file is composed of a collection of records that usually have some logical relation to one another. The record is the basic unit of information read from or written to a file by an application program.

Chapter 7 details the system I/O philosophy and includes the system calls available for file handling. The data transfer calls have an important relationship to the physical structure of a tape file and will be reviewed briefly.

The system provides two major means for specifying data for transfer from/to a file: read/write line (RDL, WRL) and read/write sequential (RDS, WRS). The first is intended for transferring ASCII information. In this scheme, a character string of arbitrary length is ended with a terminator character which is any code in the range 0:F (hex) except 9 (which is a tab). The string length is implied by the terminator's position.

The second scheme is intended for transferring non-ASCII (binary) information. In this method, string lengths are explicitly determined by a character count provided with the system call.

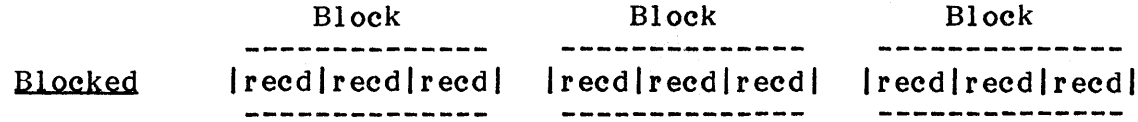
The information passed in one data transfer call is considered by the operating system to be a record.

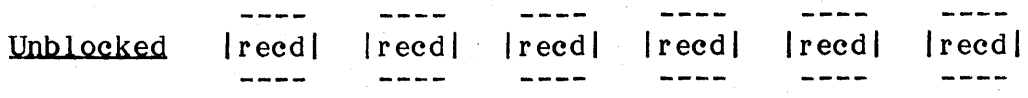
The process of grouping a number of records before writing them on a tape volume is referred to as blocking. A block is considered to be made up of the data between interrecord gaps. All blocks need not necessarily be the same size; however, a block is the minimum amount of data that may be read from or written to the tape device by the operating system.

On a magnetic tape file, records may be in one of three formats: fixed-length (F), variable-length (V), or undefined-length (U). This determines the tape mode which is defined at the time of the open call for the tape file. The maximum record and block size is 32767 characters.

FIXED-LENGTH RECORDS

The size of fixed-length (F-mode) records is the same for all records in the file. This size is declared in the record size parameter in the file open call. The tape structure resulting from this tape mode is shown below:





The blocking factor (records/block) is determined by the block size parameter declared in the file open call. For F-mode tapes, block size must be an integral multiple of the logical record size.

For data input through an RDL call, as much of the next record will be moved into the user's data area as is permitted by the maximum character count provided with the call. If the entire record was moved into the user's data area, and if room for one more character also exists, a terminator will be generated and included at the end of the data. If a terminator cannot be fit into the user's data area, an error return will occur with the system "line too long" error code (ERLTL). In any event, a return count is provided with both normal and abnormal returns which specifies the amount of data in the user's data area. If a normal return occurs, the entire record and a terminator appear in the user's data area. Regardless of the amount of data which fits in the user's data area, the entire record will be "consumed" by the read call so that the next read call will begin with the first character of the subsequent record. Data missing due to a line too long error may be obtained only by rereading the record with a larger data area.

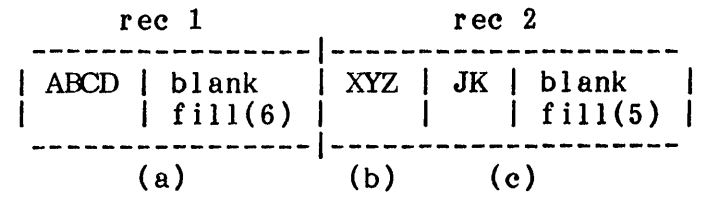
For data input using an RDS call, the comments above apply except that no terminator will be generated. In both cases, the system will not attempt to trim trailing blanks or perform any other editing, but will move a record intact (subject to size) into the user's data area.

Even though all records in the file are the same size, the data provided in a write call (WRL, WRS) may be of arbitrary length. This data string will be truncated or padded with ASCII blanks in order to match the declared file record size. The terminator associated with the data on a WRL call will not be written on tape but merely serves to imply the data length to the system. For a WRS call, the count is interpreted as the actual data count, then truncated or padded as necessary to construct the record.

For F-mode tapes, each standard WRS, WRL call causes a complete logical record to be written to the tape file. However, the WRS call does allow a non-standard form of the call to permit a single record to be built from multiple WRS calls. This non-standard WRS call applies only to tape files. The non-standard form requires that the character count be specified as a negative value. This will cause the data provided to be added to the current record without "closing out" this record. This may be done repeatedly as necessary to build up a single logical record. The maximum data accumulated is, of course, still subject to the declared logical record size. Data over this limit will be lost. A standard WRS, WRL call will close out the logical record in progress after such a series of non-standard WRS calls. Such a call need not specify any data; the current record will be padded if necessary by the standard call.

The following diagrams illustrate the standard and non-standard use of the WRS call for a tape file. This is applicable only to F-mode tapes; in the example, the logical record size is 10 characters and the block size is 20 characters.

Block 1

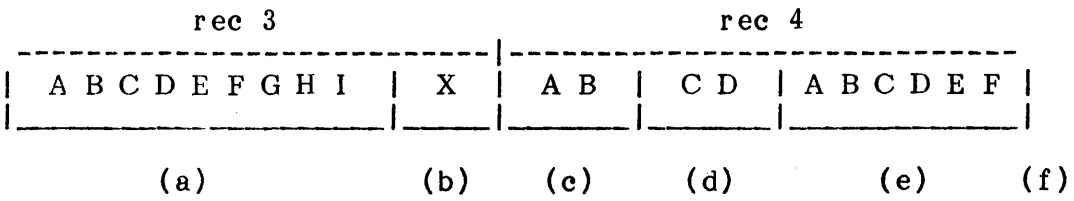


(a) 4 characters from standard WRL (or WRS) with automatic blank fill of record.

(b) 3 characters from first non-standard WRS call.

(c) 2 characters from second standard WRS (or WRL) call.

Block 2



(a) 9 characters from non-standard WRS call.

(b) 5 characters from standard WRL, WRS call; adds 1 character, loses 4 and completes current record.

(c) 2 characters from non-standard WRS.

(d) 2 characters from non-standard WRS.

(e) 10 characters from non-standard WRS call; adds 5, loses 5.

(f) Standard WRS (WRL) call closes out record.

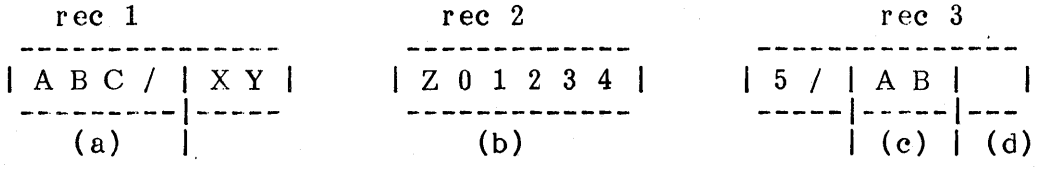
Since an output file on tape may be shared by multiple write processes, and since, for such a shared file, data interleaving can occur only on a logical record level, the use of this non-standard WRS call can effectively lock out other write processes which are sharing the same output tape file. This can happen if a write process outputs a partial record with a non-standard WRS call. Until the record is completely filled, any other write process attempting to write to the tape file will be forced to wait (suspended in the write call). Other write processes may proceed only when data in the file is on a record boundary. For standard WRS, WRL calls, this is the condition after every call. It is possible for other write processes to proceed even if one process is using non-standard calls, provided that

enough data has been supplied to fill the record (even though the non-standard write process has not closed the record with a standard call) or if the non-standard write process issues a close call on the tape file. A file close call always fills and closes out any partial record.

VARIABLE-LENGTH RECORDS

In this mode, records of arbitrary sizes will appear on tape grouped together into fixed size blocks. This tape structure provides an efficient method for writing both ASCII and binary information. It is intended primarily for use only within System II since the tape structure does not observe any industry conventions. For data written with WRS calls, no control information is added into the file; however, for (ASCII) data written with WRL calls, the terminator will be included to allow deblocking on an RDL call. Input deblocking for RDS calls is based on the specified character count provided with the call.

Data is accumulated until a block of the declared size has been filled; this block is then written on the tape volume. A record may span tape blocks. The following diagram illustrates V-mode tape structure with six character blocks where "/" is a terminator:



(a) 3 data characters and terminator output via WRL call.

(b) 9 characters and terminator output via WRL call, spanning three blocks on tape.

(c) 2 characters output via WRS call.

(d) Available space in current block.

When data input is requested (via RDS, RDL) from a V-mode tape file, a user data area and its size are specified with the call. An RDS call will provide the requested count of characters in the specified data area unless an abnormal return occurs (e.g., tape error, end-of-file).

An RDL call will similarly transfer data into the user data area subject to the specified count as a maximum, but each character is inspected after it is moved. The data transfer will stop before the count is exhausted if a terminator is detected; this is the normal mode of operation and will result in a normal return from the call. If the count is exhausted before a terminator character is encountered, an error return will occur with the system "line too long" error code (ERLTL) but the user data area will be filled with data. A subsequent input call will proceed with the next character in the file; no data will be lost.

For V-mode tape files being shared among multiple write processes, data interleaving in the file can occur only on a record boundary.

For V- or F-mode output files, the block size determines the amount of data written to tape in each physical block (except for a possible short block at file close). For input files, certain flexibility is available for processing V- or F-mode tapes whose block size is not known exactly. If an inadequate block size is declared, data will be lost without the system knowing; unpredictable data transfer may result due to missing terminators in V-mode tapes for instance. When there is uncertainty, a definite upper bound on the actual tape block size should be declared; this will avoid lost data. In this case for V-mode tapes, all data transfer and tape functions will operate properly. For F-mode tapes, only the record skip function will operate improperly since it depends on the blocking factor which is determined from the declared block size. Otherwise, the tape file may be processed normally. In these circumstances, it might be best to declare U-mode or use driver level support to allow the block to be inspected more closely.

UNDEFINED-LENGTH RECORDS

There is no record blocking performed by the system for U-mode magnetic tape files. Each read/write call will cause a physical tape block to be input/output. Furthermore, actual device data transfer occurs into or out of the user data area. For this mode, the terms record and block are synonymous.

For data input using an RDL call, as much of the next record will be moved into the user's data area as is permitted by the maximum character count provided with the call. If room for one more character also exists, a terminator will be generated and included at the end of the data. If a terminator cannot be fit into the user's data area, an error return will occur with the system "line too long" error code (ERLTL). In any event, a return count is provided with both normal and abnormal returns which specifies the amount of data in the user's data area. If a normal return occurs, the entire record and a terminator appear in the user's data area. Regardless of the amount of data which fits in the user's data area, the entire record will be "consumed" by the read call so that the next read call will begin with the first character of the subsequent record. Data missing due to a "line too long" error may be obtained only by rereading the record with a larger data area.

For data input using an RDS call, a single tape block will be read into the user data area, subject to the maximum data count specified with the call.

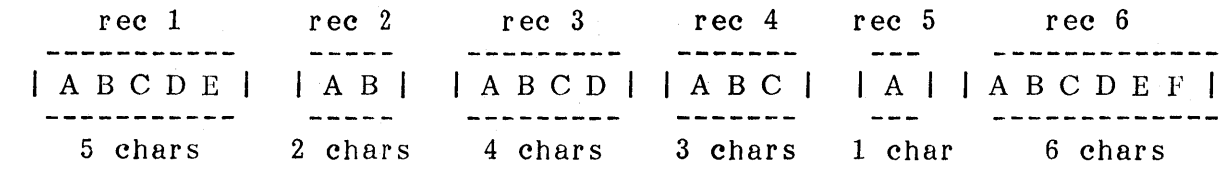
To insure that no data is lost due to an insufficiently sized data area, a size should be established that is at least one character larger than the largest anticipated tape block. Only if the returned data read count is equal to the maximum is there uncertainty as to whether the block was truncated or fit exactly.

For data output via a WRL call on a U-mode tape file, a terminator search will occur to determine the length of the record. The terminator itself is not included as part of the data. Once the data length is determined it will be written directly to tape from the user's data area.

The output process resulting from a WRS call differs only in that the data length is specified as a count instead of implied via a terminator.

If translation was selected at file open, for U-mode tapes, this translation will occur in place in the user's data area.

This diagram illustrates a possible U-mode tape structure:



Each record (block) would be read/written by a single data transfer call.

ABNORMAL CONDITIONS

The data transfer calls all provide an error return for use when an abnormal condition occurs. In general, a return data count is always provided on both read and write calls resulting in either normal or abnormal returns. An error code is specified in AC0 to indicate the nature of the abnormal condition. Several possible error codes that may occur are itemized below.

ERTIO - TAPE I/O ERROR

This error code is reserved for a class of errors associated with the tape device itself. When it occurs, the contents of UFST in the UFT must be inspected to determine the specific nature of the problem. Typical causes include not ready, offline, parity error, channel overrun and write protect. The UFT description in this section includes more detail about this status word.

EREOF - END OF FILE

This error code occurs when a file mark is encountered on tape while trying to read data to satisfy an input request. After such an encounter, the file is left positioned down tape from the file mark. A subsequent read would encounter any block immediately following the file mark. For V-mode tape files, data may have been transferred into the user data area even though an end-of-file return occurs; the return count should be inspected. (To write a file mark, see Sequential Functions in this section.)

EREOT - END OF TAPE MARK

The operating system does not consider the EOT mark on a tape volume to be sufficient reason to abort a data transfer request. This error return indicates that the requested data transfer occurred properly but that the tape volume is currently positioned beyond the EOT mark on tape. Any other error condition will take precedence over this one.

For write processes, it is suggested that they cease writing on the current volume as quickly as possible, yet writing some logical end of volume indication (possibly a simple file mark). The operating system will take no action to prevent writing off the tape volume.

A read process needs to know the rules used when the tape was generated. The operating system will allow read requests to continue (each with an EREOT return) and presumes the read process can avoid reading off the end

SEQUENTIAL FUNCTIONS

Four additional operations are provided by the operating system specifically for the support of files on magnetic tape.

WRITE FILE MARKS (MTSQWE)

This function is restricted to output tape files (opened via an OPNW call).

```
CALL
.WORD MTSQWE
error return
normal return
```

Entry Parameters: AC0 = Number of file marks.
AC2 = File UFT address.

Exit Parameters: AC2, AC3 = Restored.

For U-mode tapes where there is no blocking, the specified number of file marks will be written at the current tape position.

For V and F-mode tapes, if any data output has occurred since the open, there will be data in memory that has not yet been written to tape. This function will cause such data (possibly a short block) to be written to tape followed by the specified number of file marks.

When this function has completed, the new tape position will be immediately downtape from the last file mark, and the file will be available for continued use.

For shared tape files, this function can proceed only if the data in the file is on a record boundary. This is always true for U- and V-mode tapes as long as no write process has an incompleted output call pending. For F-mode tape files, the only exception is if non-standard WRS calls are being

used to build a record. In this case, if the file contains an incomplete record, it will automatically be filled and the block written followed by the requested file marks. The file is then on a record boundary and available to other write processes.

REWIND TAPE VOLUME (MTSQRW)

This function may be employed on either input or output files.

```
CALL
.WORD MTSQRW
error return
normal return
```

Entry Parameters: AC2 = File UFT address.

Exit Parameters: AC3 = Restored.
AC2 = Changed.

A rewind operation will be initiated on the corresponding tape drive. A subsequent read request will process data from the tape volume load point.

A more general rewind function (RWND) is available. It will rewind either an input disk file or tape file.

SKIP FILE MARKS (MTSQSF)

This function is restricted to input tape files (opened via an OPNR call).

```
CALL
.WORD MTSQSF
error return
normal return
```

Entry Parameters: AC0 = Signed file mark count.
AC2 = File UFT address.

Exit Parameters: AC2, AC3 = Restored.

A skip operation will be initiated on the corresponding tape drive. The direction of tape motion is determined by the sign of the count specified in AC0; a positive sign causes forward tape motion while a negative sign causes reverse tape motion. Motion will continue until the specified number of file marks has been encountered. For this situation, a normal return will occur and the file will be positioned just beyond (in the direction of tape motion) the last encountered file mark. For reverse motion the load point will also terminate the skip operation. A normal return will occur if only one more file mark is needed to terminate when the load point is encountered; the file will be positioned at the load point. If more than one file mark is needed, a system "tape load point" error code (ERTLP) will be returned via the abnormal return; again the file will be positioned at

load point. The end-of-tape indicator will also terminate a skip operation in either direction and will return abnormally with the system "end of tape" error code (EREOT) in AC0.

SKIP LOGICAL RECORDS (MTSQSR)

This function is restricted to input tape files (opened via an OPNR call).

```
CALL
.WORD MTSQSR
error return
normal return
```

Entry Parameters: AC0 = Signed record count.
AC2 = File UFT address.

Normal Exit: AC0, AC1, AC3 = Changed.
AC2 = Unchanged.

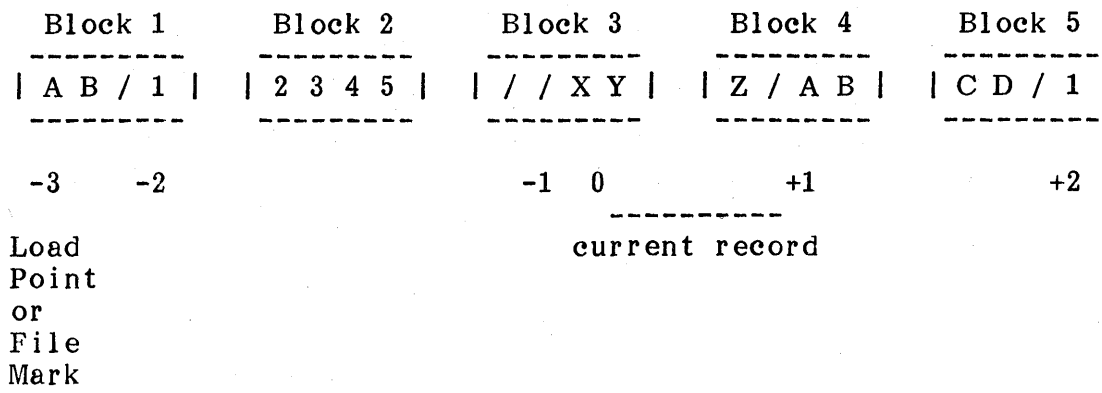
Error Exit: AC0 = System error code.
AC2 = Unchanged.
AC1, AC3 = Changed.

The tape file will be repositioned relative to the current position according to the signed record count specified in AC0. A positive sign indicates forward positioning while a negative sign causes reverse positioning.

For U-mode tapes, since each logical record is a physical block, this function is simply a skip data block operation.

For V-mode tapes, since each logical record can vary in length, and since the record boundaries are implied by terminator characters embedded in the file (or if not, the record boundaries cannot be deduced from the file alone), record skipping proceeds by terminator search. The file is examined character by character for terminators in order to establish the desired position. It is the user's responsibility to ensure that his data file contains terminators. This essentially means that a binary file (written via WRS calls) in V-mode does not have a well-defined record skip function.

For ASCII data with terminators, it is possible for the current read position to be in the middle of a record rather than always on a boundary. This would result after a "line too long" situation. As a result, a zero record count is useful as well as a non-zero value. Zero should be interpreted to mean that the file position needs to be reestablished at the first character of the current record. A record count of -1 causes the file to be repositioned at the beginning of the last full record read while +1 causes positioning to the beginning of the next full record in the file. Consider the following diagram where "/" is a terminator:



The signed counts show the corresponding new position which would be established by the skip record function if the current file position were anywhere inside the indicated record.

For a skip in the forward direction, a file mark or an end-of-tape indicator will terminate the operation with the EREOF or EREOT error codes, respectively. For a file mark, the tape volume will be left positioned beyond the file mark.

For a skip in the reverse direction, either load point or a file mark will terminate the operation. A normal return will occur for either if only one further terminator is required to terminate anyway. If load point causes this situation, the file will be left at load point; however, if a file mark is encountered leading to a normal return, the file will be repositioned back over the file mark on the side from which it was approached. The diagram above illustrates the situation. If either the load point or a file mark precede block 1, the -3 skip operation will end at the position shown in spite of no explicit terminator. However a -4 skip request would end in an error return with either the EREOF or ERTLP error codes. If the -4 ended on load point, the file position would be the same as the -3 position (indeed all higher negative skip counts would position the same). On the other hand, a file mark termination for a -4 (and higher negative) skip would position on the other side of the file mark (to the left in the diagram).

For F-mode tapes, no searching is required to determine the desired position. Since the logical record size and the blocking factor are known from their open-time declaration, the new position can be calculated and moved to directly. For a skip to a position in the current block, no tape operation will occur.

In the course of skipping records on a F-mode tape file, if a file mark, end-of-tape indicator or load point is encountered, the skip function will use the error return with the corresponding system error code (EREOF, EREOT or ERTLP, respectively). For a file mark, the file will be left positioned beyond the file mark in the direction of tape motion.

ABNORMAL CONDITIONS

Several general error conditions can occur for these functions:

ERICL - ILLEGAL CALL

These functions will return this system error code if an input file function (skip or rewind) is called to operate on an output file or vice versa. All four functions will error return with this code if OPEN was used to open the file since these are sequential I/O functions and only OPNR and OPNW imply sequential support.

ERFAP - FILE ATTRIBUTES PROHIBIT

If the file represented by the specified UFT is not on a magnetic tape device, these functions are not applicable, and this error code will be returned.

ERTIO - TAPE I/O ERROR

If a device-related abnormal condition occurs, this error code will be returned. Further investigation of the status word UFST in the UFT is required to determine the specific problem.

ERTPE - TAPE PARAMETER ERROR

For F-mode tape files, the skip record function can run into an arithmetic overflow situation which will cause this system error code to be returned in AC0 via an error return. This situation cannot occur as long as the requested skip count plus the blocking factor is less than 32768.

Also, for these functions, it is possible for file mark (EREOF), end-of-tape (EREOT), and load point (ERTLP) conditions to occur. These have been discussed in the descriptions for each specific function.

SEQUENTIAL FILE CLOSE

A close file call is the last logical step in the processing of a file. This call advises the system that the file need no longer be attached to the job, and all resources supporting the open file may be released and reused. A close call may be made by the application program or left to the system, which will close all open files associated with a job at termination (normal or abnormal).

When a close call is issued for an input file on magnetic tape, the device is not physically affected and the tape volume is left at its last position.

When a close call is issued for an output file, additional cleanup activities must occur. For F-mode, if the closing write process left the file off a record boundary by use of non-standard WRS calls, the partial record will be filled with ASCII blanks. If other write processes have also opened the file, the close call terminates; otherwise, the last block, if any, will be written to tape (V and F-mode only) followed by a single file mark.

For both input and output files on tape, internal control blocks supporting this previously open file are freed for reuse, as is the tape I/O buffer area (V- and F-modes only). If the last tape file has been closed, the partition occupied by the disk resident driver file MT9DRV.SB is available for use to support other job requests. However, since the driver is sharable (hence reusable) if the partition is still intact on the next tape file open, the driver file will not be reloaded but the intact copy in memory will be used.

DRIVER CALLS

Provisions have been made to allow user calls directly into the tape driver to permit handling tape files which cannot be dealt with using the standard, device-independent sequential I/O scheme of the operating system. These calls correspond closely with the functions provided by the tape device controller.

An open call is required as the first step. This allows the tape driver program (MT9DRV.SB) to be loaded from the primary ($) disk device and also allows certain control blocks to be set up by the operating system. In order to select driver call support, the OPEN function must be used to open the tape file.

The specific form for this open call is:

```
CALL
.WORD OPEN
error return
normal return
```

Entry Parameters: AC1 = Flags.
AC0 = File name block (FNB) address.

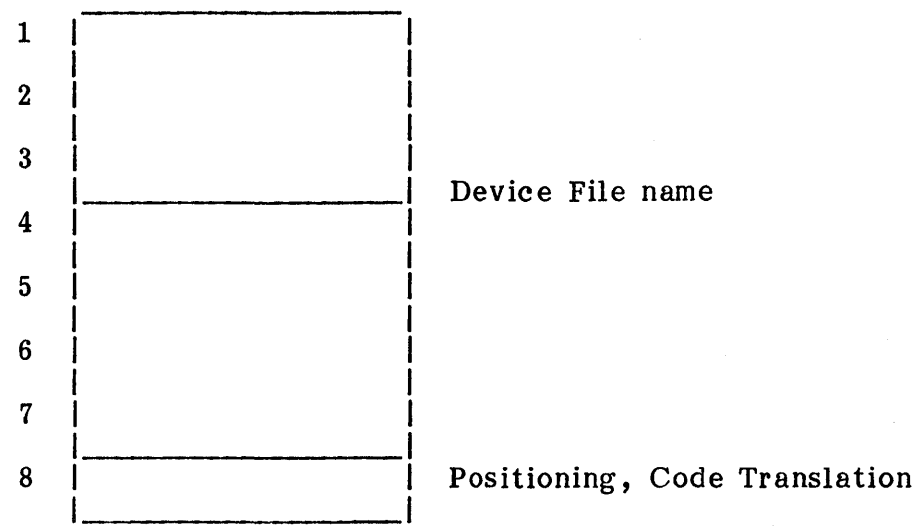
The open call provides, on a normal return, a User File Table (UFT) address in AC2. This UFT must be provided with every subsequent file service call since it identifies the file to the system. After a successful open, the UFT word UFST has a current status which indicates the state of the device. A UFT should be used only by a single task; multiple tasks should perform multiple opens so that each has its own UFT.

Flag Bit	Description
14	Write share output tape file.
8	Extended FNB specified in AC0.

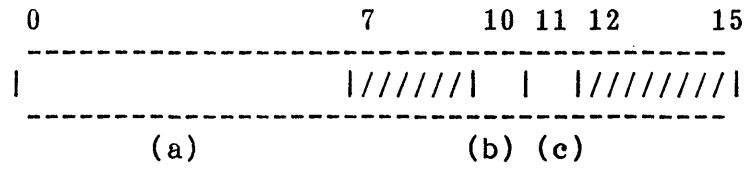
The write share bit may be set to indicate a willingness to allow other write processes to open and use the same device concurrently. If the first OPEN allows write sharing, subsequent OPNW's for write sharing are allowed. Magnetic tapes are /WS but not /RS.

Bit 8 of the flag word, when set, indicates that an extra word appears at the end of the standard 7-word FNB specified in AC0. The first seven words must contain the device file name in standard form (in words 4, 5 and 6).

An extended FNB must appear as follows:



The eighth word of the FNB (if it appears) must have the following format:



(a) Bits 0-7 0 - No skip.
n - Skip n EOF marks where n is a signed integer.
Negative implies motion toward the load point.

(b) Bit 10 Rewind before any EOF skips.

(c) Bit 11 0 - No translate (default).
1 - Translate.

An initial tape volume position may be established for an unshared file by use of the rewind and skip fields in the eighth word. Volume positioning requests on a write share open will result in an error return. The rewind bit (10) is inspected first and, if set, causes the volume to be rewound to load point. The skip field bits (0-7) contain a signed integer (-128, +127) which, if non-zero, causes the specifed number of EOF marks to be skipped. A negative count causes motion towards the load point.

The code translation option is selected by setting bit 11 in word 8 of an extended FNB.

The driver provides nine functional calls. Each must be accompanied by the UFT address which was provided by the open call. Certain functions require parameters (which will be passed in the UFT); others require none.

The general form for the driver calls is the same for each function; only the linkage name changes:

```
CALL
.WORD linkage name
error return
normal return
```

Entry Parameters: AC2 = File UFT address.

Exit Parameters: AC2, AC3 = Restored.

The UFT is a 16-word table area which identifies a particular file to the operating system and provides a central area for parameter communications and data storage. A UFT should not be shared between tasks; that is, a UFT specified in a system call should not be stored into or used in another call until the first call completes. The general structure of a magnetic tape UFT is illustrated in the following diagram:

Mag Tape UFT			Description
UFQL - 0			System UFT queue linkage
UFSF - 1			Device SFT linkage
UFOP - 2			Open flags
UFTB - 3			System TCB pointer
UFUN - 4			User scratch
UFUB - 5			Never used by driver
UFUP - 6			User scratch
UFRB - 7		*	Data buffer addr
UFRP - 8			Driver queue linkage
UFWB - 9			Driver scratch usage
UFWP - A		*	Block count for skip data
UFBS - B		*	Data count (bytes)
UFFL - C		r	Device controller status
UFST - D		r	Device operation status
UFRS - E		r	Data or skip count
UFRC - F		r	Retry count

Notation Key

* - Input Parameter
r - Return Parameter

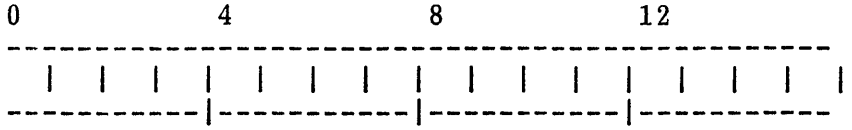
The first three locations contain system information which must be left intact. The three user slots (UFUN, UFUP, UFWP) are never used by the tape driver. The remainder of the UFT will be used when a driver function call is made except that UFWP is required only when a skip data function is requested.

The last four slots are used for return information after every driver function call. The UFFL word will contain the tape device controller status word intact (described in the product specification). The UFST word will contain the driver operation status for the last function requested. This is a combination of certain key controller status bits and some additional

information. This operation status is detailed in the next diagram. The UFRS word contains a calculated count of either the number of characters transferred for a read or write function or the count of blocks skipped for any of the skip functions. For a file mark skip, this count is an unsigned integer. The last word, UFRC, is a retry count. Only data transfer functions will be retried. Five retries will be attempted before the operation is terminated in error.

The user will vary input parameters from function to function and even call to call for the same function. In any event, the magnetic tape driver never changes the input parameter slots in the UFT (i.e., UFRB, UFWP, and UFBS).

The driver provides an operation status word in the UFST slot of the UFT after every function call. The following diagram illustrates the magnetic tape operation status word:



Bit	Description
0	Error flag.
1	Write protect indicator.
2	Load point indicator.
3	File mark indicator.
4	End-of-tape indicator.
5	Parity error.
6	Channel overrun.
7	Device offline.
8	Device not ready.
9	Timeout, device no response.
10	Maximum retries performed.
11	Error recovery in progress.
12	Retries performed.
13	Pending in-queue.
14	In progress.
15	Complete.

Bit	Explanation
0	Always set when an error return from the driver occurs. It shows that other bits are set in this status word which indicate the specific problem. Error conditions vary from function to function and are explained in the function descriptions.
1	No write enable ring is present on the tape volume mounted on the transport. No data write can occur.
2	The volume on the transport is at load point.

Bit	Explanation
3	A file mark was either written or encountered during a read or skip operation.
4	The volume on the transport is beyond the end-of-tape mark.
5	A parity error occurred during a read or write (with read after write option) operation; one of the following occurred: • Vertical parity on character • Longitudinal parity error • Cyclic redundancy check character (CRCC) error This bit will also occur if the file mark indicator is on.
6	A channel overrun condition occurred. The J100 DMA channel did not respond in time to prevent loss of data.
7	The tape transport was not online. The J100 cannot command the device in this state.
8	The tape device was not ready. The J100 cannot command the device in this state. This may be caused by device power off or disconnected.
9	The device did not respond within the allotted 5.5 second timeout period. No retries will be attempted for this condition. This will always cause a driver error return.
10	The maximum number of retries (5) were performed. When this occurs, either parity error or channel overrun should also occur. This condition will always cause a driver error return.
11	This bit is used internal to the driver to control retries and should never appear in a final operation status word.
12	At least one retry occurred. The actual retry count is contained in UFRC of the UFT.
13	This bit is used internal to the driver to indicate that the UFT has been placed on the channel queue. It should never appear in a final status word.
14	This bit is used internal to the driver to indicate the UFT is in progress on the channel. It should never appear in a final status word.
15	This bit indicates that the driver has completed the operation (either normally or abnormally) and allows the execution of the write process to continue. This bit will always appear in the final operation status word.

The following discussion describes each driver function, lists its linkage name, describes the parameters it requires and notes the possible conditions for an error return. On an error return, AC0 will always contain the system tape I/O error code (ERTIO) unless a job abort occurs. Then the system abort code (ERABT) will be used.

MTWD - WRITE DATA BLOCK

This function will cause a data block to be written on the tape device file represented by the UFT. Two parameters are required and must be specified in the UFT:

UFRB - Data location in memory

UFBS - Data byte count (32767 maximum)

The data must begin on a word boundary in memory, and the maximum block size is 32767 characters. Retries will occur if either a channel overrun or a parity error (requires read after write option) occurs. An error return will be used for any of the following conditions:

- Offline
- Not ready
- Timeout
- Max retries
- Write protect
- Load point
- File mark

If requested at open time, translation will occur in place in the write buffer. If an odd data byte count is specified, an extra character will be translated (although not written to tape).

The parameter returned in UFRS of the UFT after the write will be a byte count of the data actually written to tape.

MTRD - READ DATA BLOCK

This function will cause a data block to be read from the tape device file represented by the UFT. Two parameters are required and must be specified in the UFT:

UFRB - Memory address for data

UFBS - Maximum data byte count (32767 maximum)

On completion, any data will begin on a word boundary at the memory address specified. The actual number of characters read into memory is calculated and returned in the UFRS slot of the UFT. If requested at open time, translation will be performed in place on the actual data transferred. If the byte count is odd, an extra character will be translated.

The data count specified in the UFT will act as maximum limit on the data read. Any extra data will be lost; recovery is possible only with a reread into a larger data area. To insure no data is being lost, the data buffer should be made larger than the largest tape block. This insures that the actual data count returned in UFT slot UFRS will always be less than the maximum data count.

Retries will occur if either a channel overrun or a parity error occurs, an error return will be used for any of the following conditions:

- Offline
- Not ready
- Timeout
- Max retries
- Load point
- File mark

MTSFF - SKIP FILE MARK FORWARD

This function will cause a skip operation to be initiated in a forward direction. The skip will terminate at either a file mark or an end-of-tape indication. No parameters are required for this function.

A count of the blocks skipped will be returned in the UFT slot UFRS. This count is an unsigned integer. If the operation terminated due to a file mark, the count will include the file mark as a data block. The tape will be left positioned immediately down tape from the file mark.

No retries will be attempted for this function. An error return will be used for any of the following conditions:

- Offline
- Not ready
- Timeout
- Load point
- Channel overrun
- End-of-tape

A timeout will only occur if the device is found offline during the skip or if the device controller indicates it is not busy.

MTSFR - SKIP FILE MARK REVERSE

This function will cause a skip operation to be initiated in a reverse direction. The skip will terminate at either a file mark or a load point indication. No parameters are required for this function.

A count of the blocks skipped will be returned in the UFT slot UFRS. This count is an unsigned integer.

No retries will be attempted for this function. An error return will be used for any of the following conditions:

- Offline
- Not ready
- Timeout
- Channel overrun
- End-of-tape

A timeout will only occur if the device is found offline during the skip or if the device controller indicates it is not busy.

MTSD - SKIP DATA BLOCKS

This function will cause a skip operation over a specified number of data blocks. A signed block count must be specified:

UFWP - Signed block count

The sign of this parameter indicates the direction of tape motion: positive implies forward, negative implies reverse. The skip operation will terminate when the specified number of blocks has been skipped, or earlier if either the load point or end-of-tape mark is encountered. A file mark on tape will also terminate the operation. A positive count of the data blocks actually skipped is calculated and returned in the UFT slot UFRS. If a file mark is encountered, the count returned includes the file mark. When terminated by a file mark, the tape volume will be left positioned beyond (in the direction of motion) the file mark.

No retries will be attempted for this function. An error return will be used for any of the following conditions:

- Offline
- Not ready

- Timeout
- Load point
- File mark
- Channel overrun
- End-of-tape (if block count not satisfied)

End-of-tape will cause an error return only if the actual count of blocks skipped does not match the requested count. A timeout will occur only if the device is found offline during the skip or if the device controller indicates it is not busy.

MTWEF - WRITE FILE MARK

This function will cause a file mark to be written on the tape device file represented by the UFT. No parameters are required for this function.

No retries will be attempted for this function. An error return will be used for any of the following conditions:

- Offline
- Not ready
- Timeout
- Write protect
- Load point
- Channel overrun

MTRWD - REWIND TAPE VOLUME

This function will cause a rewind to occur on the tape device file represented by the UFT. No parameters are required for this function.

No retries will be attempted for this function. An error return will be used for any of the following conditions:

- Offline
- Not ready
- Timeout

A timeout will occur only if the device is found offline during the rewind or if the device controller indicates it is not busy.

MTERS - ERASE

This function will cause a short section of tape (about 3.75 inches) to be erased. This will permit spacing past bad sections of tape detected during a write operation. No parameters are required for this function.

No retries will be attempted. An error return will be used for any of the following conditions:

- Offline
- Not ready
- Timeout
- Write protect
- Load point
- File mark
- Parity
- Channel overrun
- End-of-tape

MTRS - READ DEVICE STATUS

This function causes no physical operation to be initiated but simply determines the device status and returns. No parameters are required and no retries will occur. An error return will be used for any of the following conditions:

- Offline
- Not ready
- Timeout
- File mark
- Parity
- Channel overrun
- End-of-tape

(THIS PAGE INTENTIONALLY BLANK)

Chapter 12

SORT

System II provides a generalized sorting program (SORTM.SB) which operates as a sharable, re-entrant subroutine.

The sorting program produces the final sorted sequence by repeatedly merging sorted subsets (runs) of the input records. With each merge these runs grow larger and fewer in number until, on the final merge, the final sorted file is output.

The sort program will allow an arbitrary number of records to be sorted into an order determined by multiple sort keys of various types. Each record to be included in the sort is obtained by a call from the sort program to a user specified reader routine. This reader routine must be constructed by the user to provide a record on each call at the memory address specified. Each record must be contiguous and may not exceed the specified record size.

In the general case, enough disk space to write each record once is required. In general, sorting time is decreased by increasing the merge order. Merge order is determined primarily by the size of sort's memory work space.

By default, the sort program obtains a work partition just big enough to use space on the scratch disk. Thus, the partition size depends on sector size and record length, measured in words. Given 12-word records, for example, these decimal approximations apply: 64 words per sector (wps) - 432 memory words; 256 wps - 1008 memory words; 512 wps - 1776 memory words. The work partition is used for I/O buffers and tables. There is a direct proportion between the number of tables and the merge order, and thus between the merge order and the size of the work partition.

If only memory, with no scratch disk, is used, the default size of the work partition is 2048 (decimal) words.

During the final merge, the complete sorted file is generated and output record by record to a user specified "writer" routine. This writer routine must be constructed by the user to accept a record and move it out of the specified buffer area prior to returning to the sort program.

The sort program can be aborted by pressing the <CANCEL> key. This causes the initiation of a cleanup operation. When the cleanup is complete, an abnormal return will be taken with the appropriate error code. This may take some time to complete, so the abnormal return will not necessarily be immediate.

The sort program saves user-task base page on entry and restores it when the reader, writer, or any user compare routine is called, and at sort termination. Also, the top eight entries in the stack are saved for the duration of the sort and are restored only at termination.

SORT CALLING SEQUENCE

The sort module, SORTM.SB, is sharable and re-entrant; it should be dynamically loaded at run-time by using the LDPRG (Load Program) call. LDPRG searches memory for a copy of SORTM.SB; only if none is present will it load the module from the current primary disk.

Upon return from LDPRG, AC1 contains the TSECT origin and the entry point of the SORTM.SB module. The calling sequence is:

```
JSR entry point of SORTM.SB
error return
normal return
```

Entry Parameter: AC3 = Address of the Sort Definition Table.

Exit Parameters: AC0 = On error return, system error code.
On normal return, changed.

AC1 = Changed.

AC2 = Changed.

AC3 = Address of Sort Definition Table.

Programming example:

```
LD      0,FNB     ;ADDRESS OF FILE NAME BLOCK FOR $SORTM.SB
LI      1,0       ;NO EXTRA TSECT SPACE
CALL
.WORD LDPRG
JMP     ERROR     ;ERROR TRYING TO LOAD
;
ST      1,SORTMP  ;SAVE ENTRY POINT
LD      3,SDTP    ;ADDRESS OF SORT DEFINITION TABLE
JSR     @SORTMP   ;START THE SORT
JMP     SERROR    ;ERROR IN SORT
 .                 ;NORMAL RETURN, SORT COMPLETE
 .
 .
```

SORT DEFINITION TABLE

A Sort Definition Table consists of the following words, as detailed on the next few pages. Recommended symbolic offsets are shown here, but they are not predefined in SYSDEF.RB.

Overall Control:

SRCSZ	Translation flag and record size
SRDR	"Reader" routine address
SWTR	"Writer" routine address
SPSZ	Work partition size
SFILNM	Scratch unit file name block address (or X'0000)
SRECT1	1st word of double-word record count
SRECT2	2nd word of double-word record count
SFILES	Number of scratch files used
SRECDS	Number of records in memory at one time
SPASS	Passes made over the records
SRUNS	Initial sorted runs created

Definition of each key:

SLKEY	Sort key flags and length
SKORG	Key displacement, in words

SCTYP Comparison-type code

SURTN Address of Compare, a user-provided routine (or X'0000)

(Further definitions follow as needed.)

The word following the last key definition:

------ X'0000, "End of Definitions" marker, indicating end of table

The word following the "End of Definitions" marker is used only if translation is requested:

STFNBA Address of FNB for string-key translation table, or X'0000 for the default table, $EBCDIC.CT

The Sort Definition Table provides all the parameters necessary to define the sort, and provides several locations which the sort program uses to communicate certain statistics back to the user. The sort-key definitions are contiguous in the table.

SRCSZ Bit 0 (the high-order bit) must be 1 to request string key translation (described later), and 0 otherwise. Bits 1 to 15 contain the record size in words; all records to be sorted must be the same size.

SRDR Entry address of the user-provided "reader" routine. It will be called repeatedly to provide each record (one per call) to be included in the sort.

SWTR Entry address of the user-provided "writer" routine. During the final merge each record in sort sequence will be output by a call to the writer routine.

SPSZ If there is no scratch disk, when SPSZ is zero a default partition of 2048 (decimal) words is obtained; when SPSZ is non-zero it specifies the number of words in the work partition to be obtained.

If there is a scratch disk, when SPSZ is zero a default merge order of 2 is used to calculate a minimum work partition, based on the scratch disk's device type; when SPSZ is between 1 and 50 (decimal) it is taken to be the merge order, and is used to determine the partition size; when SPSZ is greater than 50 it specifies the number of words for the work partition.

SFILNM When non-zero, this entry is the address of a standard 7-word File Name Block (FNB). The scratch disk is specified by the prefix part of this file name block. See Chapter 7 for the details of how the prefix specifies a disk.

If SFILNM is zero, thus specifying no scratch disk, an attempt will be made to perform the sort entirely in memory. This will be successful only if all records will fit in the work space simultaneously.

SFILES If this word contains a value between 2 and 50, that value is used as the maximum merge order. If not, a value of 50 is used.

The next six words in the Sort Definition Table are for the return of statistical parameters which describe the sort.

SRECT1
SRECT2 Two words which form a 32-bit integer quantity, giving a count of the number of records which were sorted. (SRECT1 is the high-order part of the count.)

SFILES The merge order.

SRECDS A count of the number of records that could be simultaneously held in the sort work space.

SPASS A count of the number of passes made over the pool of records.

SRUNS A count of the initially generated sorted subfiles or "runs" produced by the sort program (unsigned 16-bit integer).

The above eleven words form the basic Sort Definition Table. The balance of the table is variable in length; for each sort key to be employed, it contains a 4-word entry with the following structure:

SLKEY Bits 0-1: flags. Bits 2-15: key length in words.

SKORG Key displacement in the record (words).

SCTYP Comparison type code, described on the next page.

SURTN Address of User Compare routine; zero otherwise.

At least one sort key must be specified. The end of the table is indicated by an entry in which SLKEY contains all zeros.

A key is not considered valid unless it falls completely within the boundaries of a single record. Furthermore, SCTYP must be a legal value unless a User Compare routine is specified (SURTN nonzero).

Although SLKEY and SKORG are word-oriented, it is still possible to process character-oriented keys which do not actually coincide with word boundaries within a record.

For SCTYP codes 6 to 9, if bit 0 of SLKEY is 1, then the first eight bits of the specified key's first word will participate in neither translation nor comparison. If bit 1 of SLKEY is 1, then the last eight bits of the

key's last word are similarly excluded. Both of these SLKEY flags may be 1 at the same time. The flags are ignored for other SCTYP codes.

STANDARD COMPARISONS

If a key does not have a User Compare routine (that is, if SURTN is zero), then the sort program performs a standard comparison, as determined by the SCTYP codes listed here with typical applications.

0 Ascending, signed compare over entire key.
1 Descending.
For 64-bit decimals (chapter 14), and similar data.

2 Ascending, signed 16-bit compare on each word.
3 Descending.
For 16-bit signed integer data.

4 Ascending, logical 16-bit compare on each word.
5 Descending.
For unsigned integers and similar data.

6 Ascending, logical 8-bit compare on each character.
7 Descending.
For normal ASCII characters.

8 Ascending, logical compare on bits 1 to 7 in each character.
9 Descending.
For characters with a high-order parity bit.

STRING KEY TRANSLATION

If bit 0 of SRCSZ is 1, then the user is requesting automatic translation of all string keys (those with SCTYP codes 6 to 9). After the Reader routine passes string keys to the sort, all the characters in those keys are translated according to a specified table. The translated keys then participate in record ordering, but they are translated again, presumably to their original values, before the Writer routine receives them. (Overlapping string keys produce unpredictable results, and should be avoided.) The user thus gains the ability to specify a non-standard collating sequence for character-oriented data.

A user-supplied translation table must be a disk file with the format described in Chapter 15; STFNBA must contain the address of the file's FNB, as described in Chapter 7. The "output" section of the table is used before record sorting; the "input" section is used to restore each key's value before the Writer receives it.

If bit 0 of SRCSZ is 1, but STFNBA is X'0000, then the default table file, $EBCDIC.CT, is used. This file is included with each release of the standard utilities, and is listed in Chapter 15.

READER ROUTINE LINKAGE

Within the sort module itself, the equivalent of the following sequence is used to call a user-supplied Reader routine:

```
JSR Reader
Error return
End of input return
Normal return
```

Entry:	AC0 = Address of record buffer.
Error Exit:	AC0 = System error code. AC1-AC3 = May be changed.
End of Input Exit:	AC0-AC3 = May be changed.
Normal Exit:	AC0-AC3 = May be changed.

The Reader places the next logical record in the record buffer.

The buffer will be sized according to the record-size entry in the Sort Definition Table.

Three returns are provided:

- Record available — RTS 2 shows the specified buffer area has been filled with the next logical record.
- End of input — RTS 1 indicates that no further input will be provided. Proceed to sort records already provided.
- Error return — RTS 0 indicates an unrecoverable error has been encountered by the reader routine. The sort program will cleanup and abnormally terminate.

For an error return, the value in AC0 will be passed back as an error code when the sort program abnormally terminates.

WRITER ROUTINE LINKAGE

Within the sort module itself, the equivalent of the following sequence is used to call a user-supplied Writer routine:

```
JSR Writer
Error return
Normal return
```

Entry:	AC0 = Address of next logical record in sorted sequence.

Error Exit:	AC0 = System error code. AC1-AC3 = May be changed.
Normal Exit:	AC0-AC3 = May be changed.

The Writer processes the specified logical record. In most cases the Writer writes the information contained in the record to a sequential file or device.

When the Writer returns to the Sort program, the record buffer is reused.

Two returns are provided:

- Normal return — RTS 1 shows that the logical record provided has been transferred out of the buffer area and that "writer" is prepared to accept the next record is sequence.

- Error return — RTS 0 indicates that an unrecoverable error has been encountered by the writer routine. No further output is desired; the sort program will cleanup and abnormally terminate.

For an error return, the value in AC0 will be passed back as an error code when the sort program abnormally terminates.

COMPARE ROUTINE LINKAGE

Provisions are made in the sort key table for a user-provided Compare routine. This is done on a key-by-key basis, which allows a different Compare routine for each key if desired.

Within the sort module itself, the equivalent of the following sequence is used to call the user-supplied Compare routine:

```
JSR Compare
Record-AC0-before-record-AC1 return
Record-ordering-not-determined return
Record-AC1-before-record-AC0 return
```

The Compare routine is passed the buffer addresses for the two records to be compared. In addition, an address is passed for the particular entry in the sort key table which specifies the key to be checked.

Three returns are permitted from the Compare routine:

- Record 1 first — RTS 0 shows that the record indicated via AC0 should appear in the final sort before the record indicated via AC1. No further keys will be compared.

- Keys equal — RTS 1 shows that the ordering of the two records is not determined by the current key. Further keys will be compared to differentiate between them. If no more keys are specified, the record indicated via AC0 will appear before the record indicated via AC1.

- Record 2 first — RTS 2 shows that the record indicated via AC0 should appear in the final sort after the record indicated via AC1. No further keys will be compared.

MEMORY REQUIREMENTS

The sort program maintains all data and other working information in a system provided partition. Each sort request is allocated a work space independent from any concurrent sorts; it is this fact that allows the sort program to operate reentrantly.

The following symbols are used in the subsequent discussion to represent quantities that may vary from sort to sort.

S Scratch unit sector size (words).

R User record size (words).

W Work partition size (words).

F Fixed overhead area size (words).

M Merge order.

P Sort order (maximum number of records which can fit in the work space).

M and P are calculated from the other four quantities.

There are certain fixed assignments in the work space that exist for the duration of the sort:

- Fixed area - F — Contains pointers, counters and other parameters used by the sort program (approx. 64 words).

- Output buffer — Used to buffer data records being output to scratch on disk (S+R-1 words).

- I/O tables — 64*M words.

Then the total space required for these three items is:

F+(S+R-1)+(64*M)

There are two major sort phases. The work space is used differently in these phases. The first is the distribution phase during which all runs are written to to the scratch disk from the data records obtained from calls to the reader routine.

During the distribution phase the work space will also contain the following items in addition to the fixed assignments:

- Record buffers — Used to buffer sort records obtained from the reader routine (P buffers at R words each).
- Sort vector — Used to keep order relationships between all sort records held in memory (3*P words).

These, when combined with the fixed assignments, yield an expression for the distribution phase work area requirements:

W=F+(S+R-1)+(64*M)+(3*P)+(P*R) (Equation I)

Similarly, for the merge phase, the following items are required:

- Input buffers — For each run being merged, an input buffer of size S+R-1 words is needed.
- Sort vector — Used to keep order relationships between all runs (3*M words).

For merge phase work area requirements, we combine these two items with the fixed assignments to yield:

W=F+(S+R-1)+(64*M)+(3*M)+(M*(S+R-1)) (Equation II)

Rearranging equation II we obtain an expression for M:

M=(W-F-S-R+1)/(S+R+66)

Rearranging equation I we obtain an expression for P:

P=(W-F-S-R-(64*M)+1)/(R+3)

Consider an example:

```
W =  2048 words     work space
F =  64 words       sort fixed area
S =  256 words      scratch unit sector size
R =  30 words       data record size
```

M=(2048-64-256-30+1)/(256+30+66)=4.8

The merge order is 4.

P=(2048-64-256-30-(64*4)+1)/(30+3)=43.7

43.7 records may be held in memory at once.

For a slightly different point of view, it can be useful to know how much additional memory is required to increase the merge order by one. This turns out to be simply S+R+63 words.

Unfortunately, it is not as simple for increasing the sort order. In the worst case, a minimum of R+3 words will be required for each additional record desired. However, if the additional space allows an increase in the merge order, 67 words for I/O tables will be subtracted from the record buffer area.

As a rule of thumb, R+3 words will increase the sort order by one, while S+R+63 words will increase the merge order. The expressions for M and P should be re-evaluated to ensure the desired results.

In addition to partition work space, certain system buffer pool space is required. A sector size buffer is obtained to support disk sector allocation/deallocation. Insufficient space will cause an abnormal sort program termination.

The sort program will attempt to sort without scratch space if the scratch unit FNB pointer is zero. In order to be successful, enough work space must be allocated to simultaneously hold P+1 records when there are P records to be sorted. Since no space is required to support disk I/O the expression for P reduces to:

$$P=((W-F)/(R+3))-1 \qquad \text{(Equation III)}$$

For the earlier example:

$$P=((2048-64)/(30+3))-1=(1984/33)-1=59$$

A maximum of 59 records could be sorted without scratch file I/O.

(THIS PAGE INTENTIONALLY BLANK)

Chapter 13

SYSTEM FUNCTIONS

System II includes entry points for several function calls: date, time, and arithmetic. In addition some standard conversion routines are provided as binary modules which can be linked with user assembly language programs using RLDR.

TIME AND DATE FUNCTIONS

The time and date calls either set or retrieve the time or date in binary format. The conversion routine, DATE.RB can be used to retrieve time or date in ASCII format.

GET SYSTEM DATE (GDAT)

```
CALL
.WORD GDAT
normal return
```

Entry Parameters: None.

Exit Parameters: AC0 = Year.
AC1 = Month.
AC2 = Day.

GDAT returns the system date using a base of 1900. Thus, if AC0 = 75, the year is 1975.

GET TIME OF DAY (GTOD)

```
CALL
.WORD GTOD
normal return
```

Entry Parameters: None.

Exit Parameters: AC0 = Hours.
AC1 = Minutes.
AC2 = Seconds.

GTOD returns the current time of day.

SET SYSTEM DATE (SDAT)

```
CALL
.WORD SDAT
normal return
```

Entry Parameters: AC0 = Year.
AC1 = Month.
AC2 = Day.

Exit Parameters: AC0 = -1. Invalid date specified. System date not changed; AC1, AC2 changed.

AC0 ≠ or not equal -1. Valid date specified. System date changed; AC0, AC1 and AC2 changed.

AC3 = Unchanged.

SDAT sets the system date using 1900 as the base year. Thus, if (AC0) = 75, the year is 1975.

SET TIME OF DAY (STOD)

```
CALL
.WORD STOD
normal return
```

Entry Parameters: AC0 = Hours.
AC1 = Minutes.
AC2 = Seconds.

Exit Parameters: AC0 = -1. Invalid time specified. System time not changed; AC1, AC2 changed.

AC0 ≠ or not equal -1. Valid time specified. System time changed; AC0, AC1 and AC2 changed.

AC3 = Unchanged.

STOD resets the time of day clock.

ASCII TIME AND DATE ROUTINES (DATE.RB)

The module DATE.RB has two entry points, ADATE and ATOD. When DATE.RB is linked to an application program the date and time can be obtained in ASCII form by using JSR@ to these entry points. The entry point names must be defined as globals. For example:

```
        .GLOBL ADATE,ATOD
          .
          .
          .
GDATE:  .WORD ADATE
GTIME:  .WORD ATOD
          .
          .
          .
        JSR @GDATE
          .
          .
          .
        JSR @GTIME
```

ASCII DATE (ADATE)

JSR ADATE return

Entry Parameters: AC1 = Byte offset into buffer.
AC2 = Address of buffer.

Exit Parameters: AC0-AC3 = Unchanged.

ADATE returns an ASCII date in the format YY/MM/DD beginning at byte AC1 in the specified buffer. The date returned is packed two characters per word.

ASCII TIME OF DAY (ATOD)

Entry Parameters: AC1 = Byte offset into buffer.
AC2 = Address of buffer.

Exit Parameters: AC0-AC3 = Unchanged.

ATOD returns an ASCII time in the format HH:MM:SS beginning at byte AC1 in the specified buffer. The time is packed two characters per word.

ARITHMETIC FUNCTIONS

The ARITH system call provides access to the following arithmetic functions:

ZDEC Convert integer to decimal number.

ZIFIX	Truncate decimal number and convert to integer.
ZDNEG	Negate a decimal number.
ZDABS	Get absolute value of a decimal number.
ZFIX	Truncate decimal number, leave as decimal.
ZINT	Get largest whole decimal number not greater than input number.
ZROUND	Round to nearest whole decimal number.
ZIINT	Get largest integer not greater than input number.
ZIROUN	Round to nearest integer.
ZISGN	Give decimal result of -1 for negative number, 0 for zero, +1 for positive number.
ZIISGN	Give integer result of -1 for negative number, 0 for zero, +1 for positive number.
ZRND	Generate pseudo-random decimal number between 0 and 1.
ZMPY	Integer multiply.
ZDIV	Integer divide.
ZIDV	Integer inverse divide.
ZEXP	Integer to integer power, with integer result.
ZIEXP	Inverse (in terms of argument order) integer to integer power, with integer result.
ZDLD	Load DAC* from user area.
ZDADD	Decimal addition.
ZDSUB	Decimal subtraction.
ZDCMP	Decimal comparison, with integer result of 0 if argument and contents of DAC are equal, 1 if contents of DAC are greater, or -2 if argument is greater.
ZDMPY	Decimal multiply.

* DAC = Decimal Accumulator, described on the next page.

ZDDVD Decimal divide.

ZDIVD Decimal inverse divide.

ZDEXP Decimal to integer power.

ZDST Store contents of DAC in user area.

In order to support these functions, the system configuration must include either one of the BASIC arithmetic modules, BARITH.RB or BARSTR.RB; or ZARITH.RB, which does not support BASIC. A program that uses the ARITH call to access a function must list the name of the function as a .GLOBL symbol; then the assembly module must be linked with BDEF.RB or YIBDEF.RB, whichever defines the symbol.

The arithmetic routines operate on values in internal binary form. For conversion of ASCII decimal data to or from internal decimal form the module YICONV.RB must be linked to the user program. That module is described later in this chapter.

ARITHMETIC PROCESSOR CALL (ARITH)

```
CALL
.WORD ARITH
.WORD (name of arithmetic function)
error return  (Always present. NOP for some calls.)
normal return
```

DAC, the Decimal Accumulator, is a 40-word block used during arithmetic operations. The first four words contain one argument for operations with decimal arguments.

Entry Parameters: AC2 = Decimal Accumulator address.

AC0 = For operations with integer arguments, contains value of one argument; unused for decimal arguments.

AC1 = For operations with two arguments, either integer or decimal, contains the address of the second argument. Note: The arithmetic processor was developed to support BASIC, and the convention of using AC0 for integer values and AC1 for address pointers derives from code generation procedures in the BASIC compiler.

AC3 = Saved and restored.

Exit Parameters: AC2-AC3 = Unchanged.

AC0 = For operations with integer results, contains integer value.

On error returns, contains error number.

Unchanged for calls which return decimal result.

AC1 = For ZDIV and ZIDV calls, contains remainder of integer division.

For all other calls, unchanged.

DAC = For operations with decimal result, contains result. If error return, set to largest decimal number with appropriate sign.

Error Codes: ERDVEX Division by zero.
EROVFL Arithmetic overflow.

NUMERIC REPRESENTATION

Integers can range from -32,768 to 32,767. An integer is contained in a single word in signed two's-complement form.

A decimal number requires four words of storage. Decimal numbers are represented internally by a signed two's complement 4-word binary integer, with an implied scale factor of 10**-9 (10 to the negative 9th). Thus, decimal numbers can range from -9,223,372,036.854775808 to 9,223,372,036.854775807. The internal representation of decimal 1.0 in hex is 0000 0000 3B9A CA00.

Addition and subtraction of decimal numbers are performed as four-word binary operations. Multiplication is also performed as a four-word binary operation, then scaled by dividing by 10**9 (X'3B9A CA00). Division is done by first scaling the dividend (multiplying by 10**9), then performing a four-word binary division.

ARITH ROUTINE DESCRIPTIONS

ZDEC - Convert Integer to Decimal

```
CALL
.WORD ARITH
.WORD ZDEC
error return
normal return
```

Entry Parameters: AC0 = Integer value.
AC2 = Address of DAC block.

Exit Parameters: DAC = Decimal value. Registers unchanged.

ZIFIX - Truncate Decimal Number to Integer

```
CALL
.WORD ARITH
.WORD ZIFIX
error return
normal return
```

Entry Parameters: DAC = Decimal value.
AC2 = Address of DAC block.

Exit Parameters: AC0 = Integer result. Other registers unchanged.
AC0 = EROVFL if error overflow.

ZDNEG - DAC = -DAC

```
CALL
.WORD ARITH
.WORD ZDNEG
error return
normal return
```

Entry Parameters: DAC = Decimal number.
AC2 = Address of DAC block.

Exit Parameters: DAC = Decimal result. Registers unchanged.

AC0 = EROVFL if error; input value is largest negative number, which has no positive counterpart.

ZDABS - DAC = |DAC|

```
CALL
.WORD ARITH
.WORD ZDABS
error return
normal return
```

Entry Parameters: DAC = Decimal number.
AC2 = Address of DAC block.

Exit Parameters: DAC = Result. Registers unchanged.

Error Codes: See ZDNEG.

ZFIX - Truncate to Whole Number

```
CALL
.WORD ARITH
.WORD ZFIX
error return
normal return
```

Entry Parameters: DAC = Decimal number.
AC2 = Address of DAC block.

Exit Parameters: DAC = Decimal result. Registers unchanged.

ZINT - Greatest Whole Number Not Greater Than Value

```
CALL
.WORD ARITH
.WORD ZINT
error return
normal return
```

Entry Parameters: DAC = Decimal value.
AC2 = Address of DAC block.

Exit Parameters: DAC = Decimal whole number. Registers unchanged.

Error Codes: See ZDNEG.

ZROUND - Round to Nearest Whole Number

```
CALL
.WORD ARITH
.WORD ZROUND
error return
normal return
```

Entry Parameters: DAC = Decimal value.
AC2 = Address of DAC block.

Exit Parameters: DAC = Decimal whole number. Registers unchanged.
AC0 = EROVFL if error.

ZIINT - Greatest Integer Not Greater Than Value

```
CALL
.WORD ARITH
.WORD ZIINT
error return
normal return
```

Entry Parameters: DAC = Decimal value.
AC2 = Address of DAC block.

Exit Parameters: AC0 = Integer result. Other registers unchanged.
AC0 = EROVFL if error.

ZIROUND - Round to Nearest Integer

```
CALL
.WORD ARITH
.WORD ZIROUN
error return
normal return
```

Entry Parameters: DAC = Decimal value.
AC2 = Address of DAC block.

Exit Parameters: AC0 = Integer result. Other registers unchanged.
AC0 = EROVFL if error.

ZISGN - Indicate Sign of Argument (Decimal)

```
CALL
.WORD ARITH
.WORD ZISGN
error return
normal return
```

Entry Parameters: DAC = Decimal value.
AC2 = Address of DAC block.

Exit Parameters: AC0 = -1 if DAC is negative.
AC0 = 0 if DAC is zero.
AC0 = 1 if DAC is greater than zero. Other registers unchanged.

ZIISGN - Indicate Sign of Integer Argument

```
CALL
.WORD ARITH
.WORD ZIISGN
error return
normal return
```

Entry Parameters: AC0 = Integer value.
AC2 = Address of DAC block.

Exit Parameters: AC0 = -1 if entry value negative.
AC0 = 0 if entry value zero.
AC0 = 1 if entry value greater than zero. Other registers unchanged.

ZRND - Generate Pseudo-Random Number

```
CALL
.WORD ARITH
.WORD ZRND
error return
normal return
```

Entry Parameters: AC0 = Integer value (see Method paragraph below).
AC2 = Address of DAC block.

Exit Parameters: DAC = Pseudo-random number in range 0 to 1.

Method: If value in AC0 is not zero, a specific number which is a function of (AC0) is returned. The value in AC0 will also generate an integer "seed" stored in the 32nd word of DAC block. If the value in AC0 is zero, the current seed is used to generate a new integer seed and a new decimal result. A sequence of 65,536 calls to ZRND, each with (AC0)=0, will generate 65,536 different decimal numbers in the range 0 to 1, in a pseudo-random order. After 65,536 calls, the cycle repeats.

The pseudo-random numbers generated by this process are not a "dense" set in the J100 decimal number representation. If the numbers are sorted they will differ by an interval of approximately .000015.

ZMPY - Integer Multiply

```
CALL
.WORD ARITH
.WORD ZMPY
error return
normal return
```

Entry Parameters: AC0 = Integer value.
AC1 = Address of integer value.
AC2 = Address of DAC block.

Exit Parameters: AC0 = Integer result. Other registers unchanged.
AC0 = EROVFL if error.

ZDIV - Integer Divide

```
CALL
.WORD ARITH
.WORD ZDIV
error return
normal return
```

Entry Parameters: AC0 = Value of numerator.
AC1 = Address of denominator.
AC2 = Address of DAC block.

Exit Parameters: AC0 = Quotient.
AC1 = Positive remainder. Other registers unchanged.
AC0 = ERDVEX if error.

ZIDV - Integer Inverse Divide

```
CALL
.WORD ARITH
.WORD ZIDV
error return
normal return
```

Entry Parameters: AC0 = Value of denominator.
AC1 = Address of numerator.
AC2 = Address of DAC block.

Exit Parameters: See ZDIV.

Error Codes: See ZDIV.

ZEXP - Integer Raised to Integer Power

```
CALL
.WORD ARITH
.WORD ZEXP
error return
normal return
```

Entry Parameters: AC0 = Value of integer base.
AC1 = Address of integer exponent.
AC2 = Address of DAC block.

Exit Parameters: AC0 = Integer result. Other registers unchanged.
AC0 = EROVFL if error.

ZIEXP - Integer Raised to Integer Power

```
CALL
.WORD ARITH
.WORD ZIEXP
error return
normal return
```

Entry Parameters: AC0 = Value of integer exponent.
AC1 = Address of integer base.
AC2 = Address of DAC block.

Exit Parameters: See ZEXP.

Note: Inverse argument order from ZEXP.

ZDLD - Load DAC (Words 0-3) From User Area

```
CALL
.WORD ARITH
.WORD ZDLD
error return
normal return
```

Entry Parameters: AC1 = Address of user area.
AC2 = Address of DAC block.

Exit Parameters: DAC = Decimal value. Registers unchanged.

ZDADD - Decimal Addition

```
CALL
.WORD ARITH
.WORD ZDADD
error return
normal return
```

Entry Parameters: AC1 = Address of decimal value to be added to DAC.
AC2 = Address of DAC block.
DAC = Decimal value.

Exit Parameters: DAC = Sum of decimal values. Registers unchanged.
AC0 = EROVFL if error.

ZDSUB - Decimal Subtraction

```
CALL
.WORD ARITH
.WORD ZDSUB
error return
normal return
```

Entry Parameters: AC1 = Address of value to be subtracted from DAC.
AC2 = Address of DAC block.
DAC = Decimal value.

Exit Parameters: DAC = DAC -AC1. Registers unchanged.
AC0 = EROVFL if error.

ZDCMP - Decimal Comparison

```
CALL
.WORD ARITH
.WORD ZDCMP
error return
normal return
```

Entry Parameters: AC1 = Address of value to be compared with DAC.
AC2 = Address of DAC block.
DAC = Decimal value.

Exit Parameters: AC0 = 0 if DAC = AC1.
AC0 = -2 if DAC less than AC1.
AC0 = 1 if DAC greater than AC1. Other registers unchanged.

ZDMPY - Decimal Multiply

```
CALL
.WORD ARITH
.WORD ZDMPY
error return
normal return
```

Entry Parameters: AC1 = Address of decimal value.
AC2 = Address of DAC block.
DAC = Decimal value.

Exit Parameters: DAC = DAC*AC1. Registers unchanged.
AC0 = EROVFL if error.

ZDDVD - Decimal Divide

```
CALL
.WORD ARITH
.WORD ZDDVD
error return
normal return
```

Entry Parameters: AC1 = Address of denominator.
AC2 = Address of DAC block.
DAC = Numerator.

Exit Parameters: DAC = DAC/AC1. Registers unchanged.
AC0 = ERDVEX (divide by zero) if error.
AC0 = EROVFL if error.

ZDIVD - Inverse Decimal Divide

```
CALL
.WORD ARITH
.WORD ZDIVD
error return
normal return
```

Entry Parameters: AC1 = Address of numerator.
AC2 = Address of DAC block.
DAC = Denominator.

Exit Parameters: DAC = AC1/DAC. Registers unchanged.

Error Codes: See ZDDVD

ZDEXP - Decimal to an Integer Power

```
CALL
.WORD ARITH
.WORD ZDEXP
error return
normal return
```

Entry Parameters: AC0 = Integer exponent.
AC2 = Address of DAC block.
DAC = Decimal base value.

Exit Parameters: DAC = DAC**AC0. Registers unchanged.
AC0 = EROVFL if error.

ZDST - Decimal Store, From DAC to User Area

```
CALL
.WORD ARITH
.WORD ZDST
error return
normal return
```

Entry Parameters: AC1 = Address of user area (four words).
AC2 = Address of DAC block.
DAC = Decimal value.

Exit Parameters: AC1 = DAC address. Registers unchanged.

ASCII TO DECIMAL CONVERSION (YICONV.RB)

The module YICONV.RB has two entry points:

CSTDEC converts a numeric ASCII string to internal decimal format.

CDECST converts an internal decimal value to an ASCII string.

When the module YICONV.RB is linked to the user program these entry points are available through JSR@ instructions as follows:

```
        .GLOBL CSTDEC,CDECST
          .
          .
          .
ASTD:   .WORD CSTDEC
ADST:   .WORD CDECST
          .
          .
          .
        JSR @ASTD    ;convert to internal form
          .
          .
          .
        JSR @ADST    ;convert to ASCII form
          .
          .
          .
```

CONVERT ASCII STRING TO INTERNAL DECIMAL (CSTDEC)

The routine, CSTDEC, converts a number from ASCII string format to a four-word internal decimal format suitable for input to the System II ARITH call. The ASCII string must be preceded by a word containing the byte count, and may contain numeric digits, a decimal point, and a + or - sign on either the left or right. Blanks and commas are ignored. Any characters other than 0-9, +, -, comma, or blank cause an error return from the conversion routine.

The module, YICONV.RB, must be linked to the application program with RLDR. The entry point is CSTDEC. The interface to the conversion routine is:

AC0 = Address of the ASCII string.

AC1 = Address of a four word area for the resulting internal decimal number.

AC2 = Address of a 16-word work area.

All four registers are restored on return.

Returns are: RTS 0 if there is an error.
RTS 1 if the conversion is successful.

CONVERT INTERNAL DECIMAL TO ASCII STRING (CDECST)

The routine CDECST converts a four-word internal decimal value to an ASCII string. The format of the resulting string is a byte count followed by the number. The byte count is always 21 and the number is a full 21-character representation with 10 integer digits, a decimal point, 9 fractional digits, and a sign on the right.

The module, YICONV.RB, must be linked to the application program with RLDR. The entry point is CDECST. Input to the conversion is:

AC0 = Address of a 12-word area to receive the ASCII result.

AC1 = Address of the four-word internal decimal value.

AC2 = Address of a 16-word work area.

All four registers are restored on return.

There is only a normal return.

Chapter 14

APPLICATION PROGRAM DEVELOPMENT

A System II application program is a program which is developed by a user to be run under the control of the System II operating system. When running as a job in the system environment, it resides in a system partition and calls on the system for supervisory, memory management and I/O support.

User programs can be written in BASIC, Data-Rite, Report-Rite, or Assembly language. The BASIC and Data-Rite compilers produce executable modules which conform to System II standards. When writing in Assembly language, the programmer must comply with the following rules to produce a module which can be loaded by System II:

- The load module must be relocatable; no absolute sections (ASECT) are allowed.
- Base sector (BSECT) memory requirements must not exceed the maximum contiguous BSECT space available at load time.
- Top sector (TSECT) memory requirements must not exceed the maximum contiguous TSECT space available at load time. This will depend on the hardware memory configuration and other currently loaded programs.
- The load module must have a start address.
- All external references must be resolved.

The relocating link loader combines multiple object modules into a single load module and resolves external references among the object modules. Several binary files are provided with the system which can be specified in the link edit process to resolve references to system-defined symbols. They require no memory at run time, and are needed only to define global symbols during the link step. They include:

SYSSYM.RB — Includes all linkage parameters for system calls (listed in Appendix A) as well as task base page symbol definitions (TPZ0 - TPZ3).

ERCODE.RB Includes all error code parameters (listed in Error Messages, V3-005).

YIBDEF.RB Includes linkage parameters for the ARITH or BDEF.RB call to arithmetic routines.

CPLNK.RB Defines symbols in tables for the command parser. (Also see Global Constants, Appendix B.)

SYSSYM.RB and either YIBDEF.RB or BDEF.RB together provide symbol definitions for calls to the system for services, while ERCODE.RB defines the error code symbols used by the system.

As explained under File Referencing Conventions, the command to load a program also causes it to execute. The command specifies the disk file or non-disk device from which a binary-formatted copy of the program can be loaded, and passes any parameters that may be required to the newly created job when execution begins.

The operating system searches all the partitions in the system for a sharable copy of the requested program. If a partition is found containing the program, a new job is created for the partition and no loading occurs.

Only if no sharable copy of the specified file is found will it be loaded. An adequately sized partition will be obtained (if possible), and the operating system will proceed to relocate the specified load module into the selected partition. It then passes control to the new job at the starting address obtained from the binary file. Alternatively, the operator can instruct the system to pass control to the DEBUGGER utility rather than the program's start address. This is done by keying the character "#" immediately before the first character of the command to load the program. That utility allows the user to make minor changes and to set break points prior to the execution of the program.

For a disk file containing a reentrant (multi-user) program, the attributes of the disk file must be declared (via the CHATR system command; see System II Utilities Manual, V2-005) to be sharable for execution. Other attributes related to extended memory may be declared with the BANK utility.

For disk files containing non-reentrant (single user) programs, multiple copies in different partitions would be built to handle multiple users.

BINARY FILE LOADER

The Load Program call LDPRG, provides for direct loading of binary program files by an application. Input to LDPRG is a file name block containing the program name.

LOAD PROGRAM (LDPRG)

```
CALL
.WORD LDPRG
error return
normal return
```

Entry Parameters: AC0 = Address of a file name block (same format as required by OPEN).

Exit Parameters:
AC0 = Partition BSECT address.
AC1 = Partition TSECT address.
AC2 = Partition number.

In a J100 or J500 without extended memory, System II searches for a sharable copy of the requested program. If one is located, the use count is incremented and the partition location is returned to the caller. In a J100 with extended memory, LDPRG must provide the requested module in the same bank as the caller. If a sharable copy is present in the same bank, it will use it. If the requested program has the "unique" attribute (see BANK utility) and a copy is in use in another bank, the request will be refused. A request to load a "user sharable" program will also be refused.

Otherwise, the system will open the file, read the title block of the program to determine its size, allocate a partition for the program, load the program, close the file, and return the partition location. If the program to be loaded has the top loading attribute (TL), it will be loaded in the highest available partition space. If it does not have this attribute, it will be loaded in the lowest available partition. The specified program must not contain any absolute sections (ASECT) nor any unresolved external reference. The auxiliary partition and program are released from the job by either the FPA system call or job termination.

SYSTEM RELEASE LEVEL

It may be useful for an application program to know what release of System II it is running under. Starting with Release 8.1, this information is available at absolute TSECT address RLVL, a global symbol defined in SYSDEF.RB (not SYSSYM.RB).

For Release r.n, the word at RLVL has this structure:

Bits 0 to 7	Reserved for flags
Bits 8 to 12	Value of r
Bits 13 to 15	Value of n

Thus, Release 8.1 is encoded in binary as:

```
0000 0000 0100 0001
```

with this interpretation:

```
0000000             Flags
01000               Value of 8
001                 Value of 1
```

Chapter 15

CODE TRANSLATION FILES

Most device drivers, and some utility programs, provide for the code translation of input and output data. This process occurs if a "code translation file" with the appropriate name and the reserved extension "CT" is present on the current primary disk. If no such file is found, then no translation is done.

For a driver, the name of the relevant file is the same as the device being opened, but with the "CT" extension suffixed. As an example, a file named $LPT1.CT would be used by the line printer driver (CDODRV.SB) for translating output to device LPT1.

A customized file should be assembled and cataloged on the primary disk under a dummy name, then renamed as needed when it must be made active or inactive. The extension "DT" ("Dummy Table") is recommended for this purpose.

The file contains a table of 16-bit words; normally, the table has two sections, each 128 words long. The first section is for output processing; the second section is for input processing, and is not required for an output-only device.

Note that a Diablo HyTerm does need an input table, even if it has no keyboard. The printer hardware will send X'06 and X'86 back to the driver; both of these must have translated values of X'06, an ACK character in ASCII. Keyboard data may be translated as desired, but each resulting value must be less than X'80.

Each 128-word section of the table contains 256 replacement values, packed two per word in ascending order by the binary value of the data to be replaced.

A programming example appears on the next page. The table would be appropriate for two-way translation between ASCII and the English-language version of EBCDIC.

To aid in the creation of similar tables, a complete ASCII reference chart follows the example.

```
.TITLE AETAB,'ASCII/EBCDIC TABLE'
;
; Output section -- the computer's internal ASCII data
; is translated into EBCDIC output.  For example, an ASCII
; space (X'20) should become an EBCDIC space (X'40).  Call the
; WORD directives Row 0 to Row F; for each row, call the 8-bit
; values Column 0 to Column F.  Now look at Row r, Column c to
; find the value which replaces X'rc.
;
;         0 1     2 3     4 5     6 7     8 9     A B     C D     E F
.WORD X'0001,X'0203,X'372D,X'2E2F,X'1605,X'250B,X'0C0D,X'0E0F ;0
.WORD X'1011,X'1213,X'3C3D,X'3226,X'1819,X'3F27,X'1C1D,X'1E1F ;1
.WORD X'405A,X'7F7B,X'5B6C,X'507D,X'4D5D,X'5C4E,X'6B60,X'4B61 ;2
.WORD X'F0F1,X'F2F3,X'F4F5,X'F6F7,X'F8F9,X'7A5E,X'4C7E,X'6E6F ;3
.WORD X'7CC1,X'C2C3,X'C4C5,X'C6C7,X'C8C9,X'D1D2,X'D3D4,X'D5D6 ;4
.WORD X'D7D8,X'D9E2,X'E3E4,X'E5E6,X'E7E8,X'E94A,X'E04F,X'5F6D ;5
.WORD X'7981,X'8283,X'8485,X'8687,X'8889,X'9192,X'9394,X'9596 ;6
.WORD X'9798,X'99A2,X'A3A4,X'A5A6,X'A7A8,X'A9C0,X'6AD0,X'A107 ;7
.WORD X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000 ;8
.WORD X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000 ;9
.WORD X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000 ;A
.WORD X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000 ;B
.WORD X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000 ;C
.WORD X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000 ;D
.WORD X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000 ;E
.WORD X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000 ;F
;
;
; Input section - EBCDIC is translated into ASCII.  For example,
; an EBCDIC asterisk, X'5C, becomes the value at Row 5, Column C,
; which is X'2A - an ASCII asterisk.
;
;         0 1     2 3     4 5     6 7     8 9     A B     C D     E F
.WORD X'0001,X'0203,X'0009,X'007F,X'0000,X'000B,X'0C0D,X'0E0F ;0
.WORD X'1011,X'1213,X'000D,X'0800,X'1819,X'0000,X'1C1D,X'1E1F ;1
.WORD X'0000,X'0000,X'000A,X'171B,X'0000,X'0000,X'0005,X'0607 ;2
.WORD X'0000,X'1600,X'0000,X'0004,X'0000,X'0000,X'1415,X'001A ;3
.WORD X'2000,X'0000,X'0000,X'0000,X'0000,X'5B2E,X'3C28,X'2B5D ;4
.WORD X'2600,X'0000,X'0000,X'0000,X'0000,X'2124,X'2A29,X'3B5E ;5
.WORD X'2D2F,X'0000,X'0000,X'0000,X'0000,X'7C2C,X'255F,X'3E3F ;6
.WORD X'0000,X'0000,X'0000,X'0000,X'0060,X'3A23,X'4027,X'3D22 ;7
.WORD X'0061,X'6263,X'6465,X'6667,X'6869,X'0000,X'0000,X'0000 ;8
.WORD X'006A,X'6B6C,X'6D6E,X'6F70,X'7172,X'0000,X'0000,X'0000 ;9
.WORD X'007E,X'7374,X'7576,X'7778,X'797A,X'0000,X'0000,X'0000 ;A
.WORD X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000,X'0000 ;B
.WORD X'7B41,X'4243,X'4445,X'4647,X'4849,X'0000,X'0000,X'0000 ;C
.WORD X'7D4A,X'4B4C,X'4D4E,X'4F50,X'5152,X'0000,X'0000,X'0000 ;D
.WORD X'5C00,X'5354,X'5556,X'5758,X'595A,X'0000,X'0000,X'0000 ;E
.WORD X'3031,X'3233,X'3435,X'3637,X'3839,X'0000,X'0000,X'00FF ;F
;
.END
```

ASCII CHART

The designation or printable character associated with each 8-bit value from X'00 to X'7F is defined by ASCII, the American Standard Code for Information Interchange. The following chart shows the decimal and hex values for these characters. The character printed as a space is decimal 32 or hex 20, designated SP in ASCII.

Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII
00	00	NUL	32	20	SP	64	40	@	96	60	`
01	01	SOH	33	21	!	65	41	A	97	61	a
02	02	STX	34	22	"	66	42	B	98	62	b
03	03	ETX	35	23	#	67	43	C	99	63	c
04	04	EOT	36	24	$	68	44	D	100	64	d
05	05	ENQ	37	25	%	69	45	E	101	65	e
06	06	ACK	38	26	&	70	46	F	102	66	f
07	07	BEL	39	27	'	71	47	G	103	67	g
08	08	BS	40	28	(	72	48	H	104	68	h
09	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

(THIS PAGE INTENTIONALLY BLANK)

Chapter 16

SYSTEM AUDIT TRAIL

The System Audit Trail is a facility to gather and retain information about system activities, including information provided by application and utility programs. Such a facility has a number of uses, some of which are listed below:

- User accounting for billing supervision.
- Application accounting for billing and supervision.
- Job accounting.
- Resource usage statistics for capacity planning.

The Audit facility consists of:

- The Audit utility.
- The Audit Trail file.
- The System calls AULG, AUWR.

The Audit utility is invoked by the Audit command line, and it can open or close an Audit Trail file as well as specify which record types are to be allowed in the Audit Trail file. Each CRT may be audited by only one file, but each Audit Trail file may audit many CRTs. There may be as many active Audit files as there are CRTs. The Audit utility is described in more detail in the System II Utilities Manual.

The Audit Trail file is a sequential binary file that is written by the Audit facility. System calls will be made by the operating system, utilities and application programs to add information to this file. The information types that will be accepted by the Audit facility and the programs or terminals that are allowed to add information are specified using the Audit utility. The Audit Trail file may be used as input to a program (e.g., Report-Rite) that will extract meaningful subsets and records for report generation (see Figure 16-8, page 16-8). The Audit Trail

file can be made secure at the user option to prevent unauthorized modification of the Audit Trail data. Data will be added to the Audit Trail even if the security level of the Audit Trail is different from the user's security level. LOGIN and LOGOFF (see System II Security) can be used to associate a user name with a terminal and this name will then be inserted by the system into the header of each audit record associated with jobs run on that terminal.

The system call AULG is executed once by an application and provides a means of associating standard system audit information with the application by inserting an application specific code in the header of each audit record generated by the system. The system call AUWR is used by the operating system, utilities and applications to write a record to the Audit Trail file. Each record written has a standard header whose format is shown in Figure 16-1.

STANDARD SYSTEM AUDIT RECORDS

System II has a number of intercept points which will write out a record to the Audit Trail file. Such system audit records are output at the following points:

- Job initiation and termination.
- File open and close.
- Error message output.

If the audit facility is active for a given terminal, then error message records will always be written to the Audit Trail file. The first two intercepts, however, may be enabled/disabled by the Audit utility on a per terminal basis.

The format of each standard audit record is shown in Figures 16-2 through 16-6.

APPLICATION CODE AND RECORD TYPE

In order to classify the data produced by the Audit facility, each record contains two identifying integers (see Figure 16-1). One is associated with an application or utility and specified by the AULG system call; the other is associated with the record and passed to the Audit facility for the AUWR system call.

Each application is given a different code number, and they should be assigned so that the codes for applications of a similar nature fall within a given range. An example scheme is shown in Figure 16-7. Similiarly, records associated with the same activity should have the same record type and record format.

The intention of such allocations is that it becomes possible to extract meaningful subsets of information held in the Audit Trail; for example, by selecting only those Audit Trail records that have a certain range of application codes and/or record types.

WORD OFFSET	FIELD TYPE	CONTENT
0	STRING	USER NAME
6	STRING	MESSAGE TERMINAL NAME
10	INTEGER	SYSTEM DATE (YEAR)
11	INTEGER	SYSTEM DATE (MONTH)
12	INTEGER	SYSTEM DATE (DAY)
13	INTEGER	SYSTEM TIME (HOUR)
14	INTEGER	SYSTEM TIME (MIN)
15	INTEGER	SYSTEM TIME (SEC)
16	INTEGER	JOB IDENTIFICATION NUMBER
17	INTEGER	APPLICATION CODE
18	INTEGER	RECORD TYPE
19	INTEGER	RECORD BODY BYTE COUNT

Note: This record header is part of every record placed in the Audit Trail file by the AUWR system call. Only the application code, record type, and record body byte count is set by the calling application. The rest of the fields are set by the AUWR system call prior to writing the record to the file.

Figure 16-1. Record Header

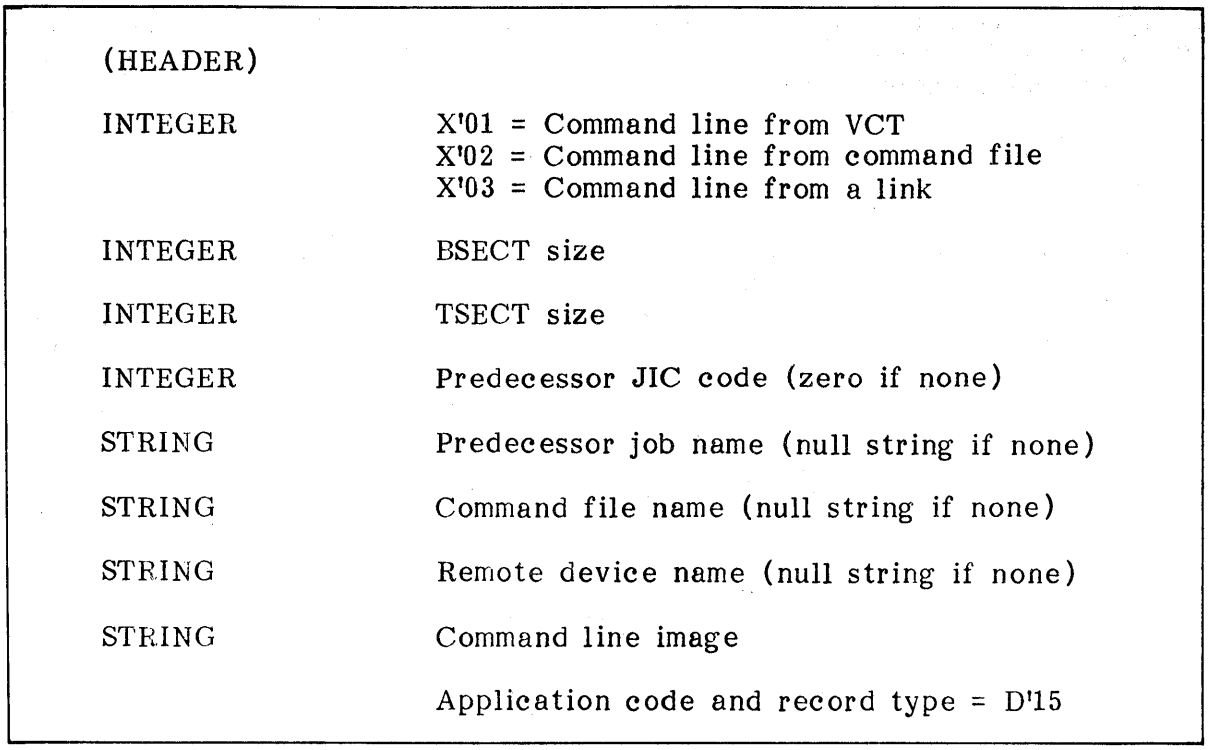


Figure 16-2. Job Initiation Record

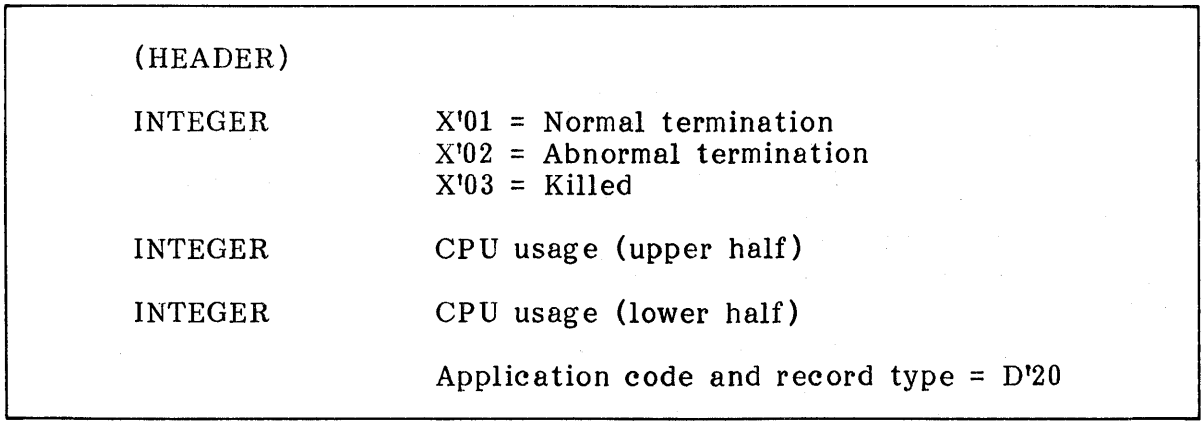


Figure 16-3. Job Termination

(HEADER)	
INTEGER	X'00 = Not a secure file X'01 = Secure file
INTEGER	File attribute flags
INTEGER	Share flags
INTEGER	Type of open
INTEGER	Use count
INTEGER	# of words in file (upper half)
INTEGER	# of words in file (lower half)
STRING	File name
	Application code and record type = D'25

Figure 16-4. File Open

(HEADER)	
INTEGER	File attribute flags
INTEGER	Use count
INTEGER	# of words in file (upper half), zero if hash
INTEGER	# of words in file (lower half), zero if hash
STRING	File name
	Application code and record type = D'30

Figure 16-5. File Close

FIELD	CONTENT
(HEADER)	SEE RECORD HEADER FORMAT FOR DETAILS
STRING	ERROR MESSAGE TEXT (NEVER NULL)
Note: This record is generated by the system error message routines whenever a system error message is generated.	

Figure 16-6. Error Message Record

A partial scheme for business applications might be:

```
.
.
.
II.  1000;10,000 - Business Applications

     A.  1000;1999 - General Ledger Programs

         1.  1000;1249 - Data Entry Programs

             a)  1000;1099 - Batch Data Entry

             b)  1100;1149 - Batch Corrections

             c)  1150;1199 - Master File Maintenance

             d)  1200;1249 - Miscellaneous Data Entry

         2.  1250;1599 - File Processing Programs

             a)  1250;1399 - Sort/Select/Merge Programs

             b)  1400;1499 - Master File Batch Updates

             c)  1500;1599 - Miscellaneous File Processing

         3.  1600;1999 - Report Generation

             a)  1600;1699 - Edit Lists and Daily Reports

             b)  1700;1799 - Weekly and Monthly Reports

             c)  1800;1899 - Quarterly and Yearly Reports

             d)  1900;1999 - Miscellaneous Reports

     B.  2000;2999 - A/R Programs

         1.  2000;2249 - Data Entry Programs

             a)  2000;2099 - Batch Data Entry Programs

             b)  2100;2149 - Batch Corrections

.
.
.
```

Figure 16-7. Example Scheme for Business Applications

	Job Run Time (Hrs)	CPU Usage (Hrs)	Per-cent of Total	Deviation From Previous Period (%)	Number of Records Processed	Deviation From Previous Period (%)	Deviation From 6 Period Average (%)
Batch A/R Data Entry	2.1	.15	2.5	+5	525	+6.0	+2.3
Batch A/P Data Entry	1.3	.08	1.3	+1	207	+2.0	+0.5
Batch Time Card Entry	5.6	1.1	18.0	+12	6,326	+15.0	+12.0
Total Batch Data Entry	9.0	1.33	23.0	+8	7,058	+12.0	+10.0
Total Batch Corrections	.7	.1	1.6	-4	52	-1.5	-2.0
A/R File Update	.75	.5	8.3	+4	525	+5.0	+2.0
A/P File Update	.41	.32	5.3	+.8	207	+1.8	+0.3

Figure 16-8. A Possible Report Format

AUDIT LOG IN (AULG)

```
CALL
.WORD AULG
not accepting return
normal return
```

Entry: AC0 = Audit application code.

Not Accepting Exit: AC0-AC3 = Unchanged.

Normal Exit: AC0-AC3 = Unchanged.

AULG sets the calling job's audit application code into the job's JCT. Each time a job calls AUWR if the job's audit application code is non-zero and the application code specified in the record is greater than or equal to 0 and less than or equal to D'49, AUWR replaces the application code specified with the job's application code. Application codes and record types 0 through D'100 are reserved for the operating system.

The purpose of the above application code replacement scheme is to simplify the post-processor's association of operating system generated records with a non-system application. For example, the accounts receivable application program might set its job's application code to identify itself. Subsequently, every operating system generated Audit Trail record, such as file opens and closes, and job termination, would be immediately identifiable by application code as associated with accounts receivable.

AULG also indicates to the caller whether or not the Audit Trail is currently accepting application records from the caller's job. The Audit Trail is currently accepting application records from the caller's job if and only if the caller's job's message terminal currently has an Audit Trail file which is accepting application records.

There is no audit application code associated with a computer terminal or with a command file. Hence, the effects of AULG last only until the calling job terminates or calls AULG again. In most if not all cases, AULG should not be used to change a job's application code from one non-zero integer to another. However, AULG may be called many times to determine whether or not the Audit Trail is currently accepting records from the calling job.

Any one word integer value may be specified as the Audit application code. AULG does not check for any error conditions.

AUDIT WRITE RECORD (AUWR)

```
CALL
.WORD AUWR
error return
normal return
```

Entry: AC0 = Address of buffer containing record.

(AC0 + D'17) = Audit application code. If this application code is >= 0 and <=D'49 and the job's application code (set by AULG) is non-zero, AUWR will set the job's application code into (AC0 + D'17).

(AC0 + D'18) = Record type.

(AC0 + D'19) = Number of bytes in record body (allowed range: 0-D'254).

(AC0 + D'20), ... = Record body.

Error Exit: AC0 = System error code.
AC1-AC3 = Unchanged.
Record header set and compressed (see below).

Normal Exit: AC0-AC3 = Unchanged.
Record header set and compressed (see below).

Error Codes:

ERDIO	Disk error.
ERDSNA	No disk space available.
ERISZ	Illegal block size.
ERBUFT	Bad UFT address.
ERICL	Illegal system call.
ERSQHF	Sequential I/O call on a hash file.
ERSQNS	Sequential I/O not supported on this device.

AUWR writes one Audit Trail record to the Audit Trail file for the job's message terminal if and only if there is such an Audit Trail file and this Audit Trail is currently accepting records with the application code specified by input parameter (AC0 + D'17). Else, AUWR does nothing and takes its normal return.

After determining whether or not to add the record to the Audit Trail, but before writing it to the Audit Trail, the application code input is replaced by the job's application code if appropriate. If the record type is greater than or equal to zero, and less than or equal to D'49, and the job's application code is non-zero, then the input application code is replaced by the job's application code.

The buffer containing the record may be in a partition or in a system buffer. Note that AUWR returns this buffer to the caller.

Note that while there may be many Audit Trail files (as many as one per computer terminal) each call to AUWR only writes the specified record to at most one Audit Trail file.

AUWR does not check the format of the record body. It is the caller's responsibility to observe the BASIC, Data-Rite, and Report-Rite data formats. Remember that the Audit Trail file is intended for processing by Report-Rite. See the Data-Rite manual for descriptions of the various Binary file data formats (e.g., Integer, String, Decimal, etc.).

AUWR does not compress or change the record body in any way.

RECORD FORMAT AS INPUT TO AUWR

Word	Type	Meaning
0-16	-	To be set by AUWR
17	Integer	Audit application code
18	Integer	Record type code
19	Integer	Number of bytes in record body (allowed range: 0 - D'254)
20-	-	Record body

RECORD FORMAT AS RETURNED BY AUWR AND AS WRITTEN TO THE AUDIT TRAIL FILE

Note that the two strings are compressed down against word 10 so that the origins of these fields depend on the string lengths. Words that are not used at the beginning of the header are not written to the Audit file.

Word	Type	Meaning
	String	User name; if necessary padded to an even number of bytes by a trailing blank.
	String	Name of job message terminal; if necessary padded to an even number of bytes by a trailing blank.
10	Integer	Year (e.g., 1980, 1981, ...)
11	Integer	Month (e.g., 1, 2, 3 ... 12)
12	Integer	Day (e.g., 1, 2, 3 ... 31)
13	Integer	Hour (e.g., 0, 1, 2 ... 23)
14	Integer	Minute (e.g., 0, 1, 2 ... 59)

Word	Type	Meaning
15	Integer	Second (e.g., 0, 1, 2 ... 59)
16	Integer	Job identification code.
17	Integer	Audit application code.
18	Integer	Record type code.
19	Integer	Number of bytes in record body (allowed range: 0-D'254).
20-	-	Record body.

Remember that the format for String type data is as follows:

Word	Contents
0	Number of bytes in string (allowed range: 0-D'254).
1	Byte #1 in left byte, byte #2 in right byte.
2	Byte #3 in left byte, byte #4 in right byte.

etc.

Appendix A

SYSTEM CALL INDEX

Listed below are the system call linkage parameters. These symbols are defined in the module SYSSYM.RB, which can be linked with the user's application program (see Chapter 14).

Symbol	Hex	Page	Function
ABT	03	6-7	Job abort.
ABTC	04	6-7	Abort control.
ARITH	7F	13-5	Arithmetic processor call.
AULG	A0	16-9	Audit log in.
AUWR	A1	16-10	Audit write record.
CEFREE	81	9-27	Erase free screen.
CEPROT	47	9-28	Erase unprotected fields.
CERALL	46	9-28	Erase entire screen.
CEROLL	82	9-29	Erase roll part.
CHTR	17	7-22	Change attributes.
CLOS	14	7-19	Close file.
CMDPAR	38	B-8	Parse command line.
CREA	10	7-9	Create a file.
CRLGTS	23	9-27	Set or read terminal status lights.
CRTLCI	68	9-26	Set or release lowercase option.

Symbol	Hex	Page	Function
CTLCRT	1F	9-10	Return to command mode.
DELHF	7E	8-15	Delete a hash file.
DELT	11	7-20	Delete a file.
DSKSP	7C	7-23	Disk space available.
FBF	0D	5-3	Free a buffer.
FMSG	30	9-20	Display system error message and file name.
FPA	40	5-5	Free a partition.
FULCRT	21	9-9	Set free screen mode.
GBF	0C	5-2	Get a buffer.
GDAT	36	13-1	Get system date.
GPA	3F	5-4	Get a partition.
GTOD	34	13-1	Get time of day.
HFADD	57	8-6	Hash file add record.
HFCLR	80	8-16	Clear a hash file.
HFDEL	5A	8-10	Hash file delete record.
HFFND	59	8-8	Hash file find record.
HFNXT	5D	8-9	Hash file find next record.
HFXCH	58	8-6	Hash file exchange record.
HLFND	69	8-13	Hash file find and lock record.
HLNXT	6A	8-14	Hash file find and lock next record.
HREAD	5B	8-11	Hash file sequential read.
LDPRG	4D	14-3	Load program.
LINK	7A	6-10	Link to secondary job.
MCRT	29	9-13	Copy data from screen.
MSG	2E	9-23	Display message.
MSRD	B5	9-23	Message read and display.

Symbol	Hex	Page	Function
MSVF	B6	9-21	Error message with file name to computer terminal.
MSVS	B7	9-19	Error message to computer terminal.
MTERS	73	11-27	Erase.
MTRD	6F	11-23	Read data block.
MTRS	74	11-27	Read device status.
MTRWD	71	11-26	Rewind tape volume.
MTSD	6E	11-25	Skip data blocks.
MTSFF	6C	11-24	Skip file mark forward.
MTSFR	6D	11-25	Skip file mark reverse.
MTSQRW	75	11-14	Rewind tape volume.
MTSQSF	76	11-14	Skip file marks.
MTSQSR	77	11-15	Skip logical records.
MTSQWE	78	11-13	Write file marks.
MTWD	70	11-23	Write data block.
MTWEF	72	11-26	Write file marks.
NSPCRT	25	9-8	Set roll mode.
OPEN	3B	7-12	Open sequential file for reading and writing.
OPNR	12	7-9	Open sequential file for reading.
OPNW	13	7-11	Open sequential file for writing.
RCRT	28	9-12	Read free screen.
RCRTB	2B	9-10	Read bottom line.
RCRTBN	2D	9-11	Read bottom line without roll.
RDL	18	7-15	Read line.
RDL	18	9-10	Read bottom line.
RDLQ	AC	7-16	Read line quickly.
RDS	1A	7-14	Read sequential.

Symbol	Hex	Page	Function
RELJ	79	6-11	Release from predecessor job.
RNAM	16	7-21	Rename a file.
ROLCRT	20	9-8	Set roll mode.
RWND	3C	7-23	Rewind sequential file.
SDAT	37	13-2	Set system date.
SMSG	2F	9-17	Display system error message.
SPBCRT	53	9-10	Set split screen boundary.
SPCRT	24	9-9	Set split screen mode.
STOD	35	13-2	Set time of day.
SUSA	0B	6-10	Suspends allowed.
SUSC	06	6-8	Suspend until location changes.
SUSN	08	6-9	Suspend until location non-zero.
SUSP	05	6-8	Suspend until next scheduler pass.
SUST	09	6-9	Test flag and suspend.
SUSX	0A	6-10	Suspends not allowed.
SUSZ	07	6-9	Suspend task until location equals zero.
TASK	01	6-6	Schedule a new task.
TEND	02	6-7	Terminate a task.
VOLID	97	7-24	Read disk volume identification.
WCRT	2A	9-16	Write screen.
WCRTB	2C	9-14	Write to bottom line of terminal.
WRL	19	7-17	Write line.
WRL	19	9-15	Write line to terminal.
WRLC	AB	7-18	Write line compressed.
WRS	1B	7-14	Write sequential.
WRS	1B	9-16	Write sequential to terminal.

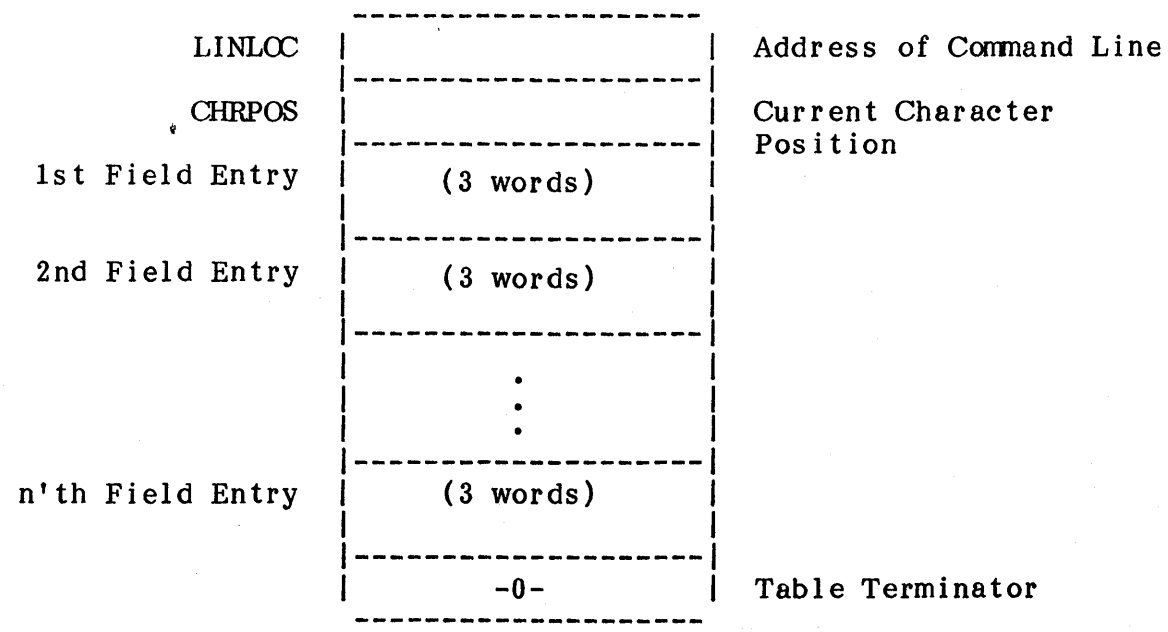
Appendix B

COMMAND LINE PARSER AND TABLE STRUCTURE

System II includes a generalized program which parses and converts command line character strings into data structures for ease of handling by application programs and other parts of the system itself.

The parser program accepts the general System II command line syntax as illustrated at the end of this appendix. However, when invoked, a table must be passed which contains pointers to the command line character string, definitions of the allowable syntactical constructs and addresses of data areas.

The syntax table is serially (not reentrantly) reusable. It has the following structure:



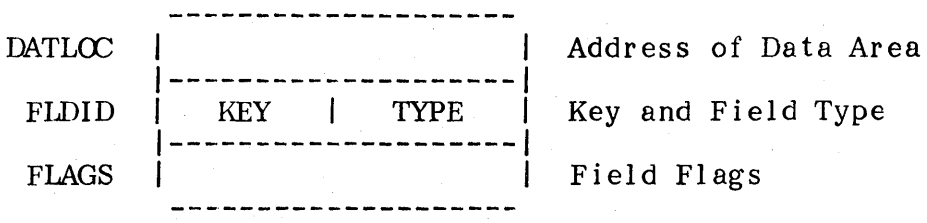
LINLOC - ADDRESS OF COMMAND LINE

This is the first word address of the command line.

CHRPOS - CURRENT CHARACTER POSITION

This is the character position at which the parser is to begin its scan. This location is maintained by the parser to always point to the next character to process or may be considered a count of consumed characters, relative to the contents of LINLOC.

It can be seen from the syntax definition that a statement consists of fields of three basic types. Each entry in the table corresponds to a "field" in the command line. Each entry in the table consists of three words and has the following structure:



DATLOC - DATA AREA LOCATION

This is the first word address of several locations in memory where data related to the command line field is to be placed (size requirements are defined in the associated field descriptions). A zero entry in this word indicates the end of the table.

If the current field is of the string type, DATLOC contains the address of a three-word control block associated with string fields.

FLDID - FIELD IDENTIFIERS

This quantity defines the permissible type of field which will correspond to this table entry. It is a two-byte field containing an eight-bit field type identifier in the right byte and a seven-bit ASCII key character or zero in the left byte.

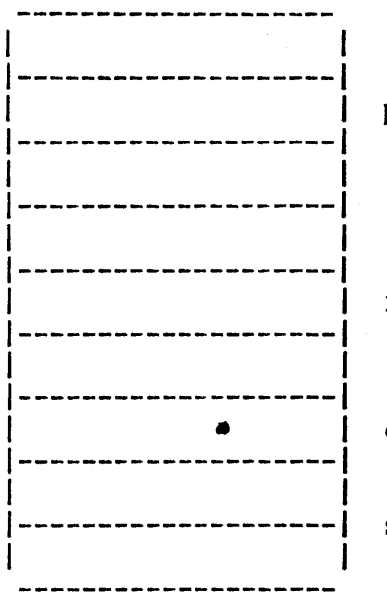
Five basic field types are permitted: file name, string, switch, numeric expression, or numeric range. In addition, the entry may require a prefix of the form "[ascii]=" on the field of specified type. The key character in the table entry indicates this. If non-zero, the field in the command line used to satisfy this table entry must be of the form "[ascii=parm]" where [ascii] is the seven-bit key character and [parm] is of the type indicated in the table entry. If the key character is zero, the field in the command line must not have a prefix "ascii=" and must conform to the type indicated.

The parser program uses the order of entries in the table to define the command line. Two exceptions to this rule exist: they are KEYED fields and switch fields. The occurrence of these causes the parser program to scan the complete table for an unused field entry which matches that found in the command line. If no match is found in the table, a "syntax" error condition is returned to the caller.

FIELD TYPES

FIELD TYPE, BIT 15 - FILE NAME REFERENCE

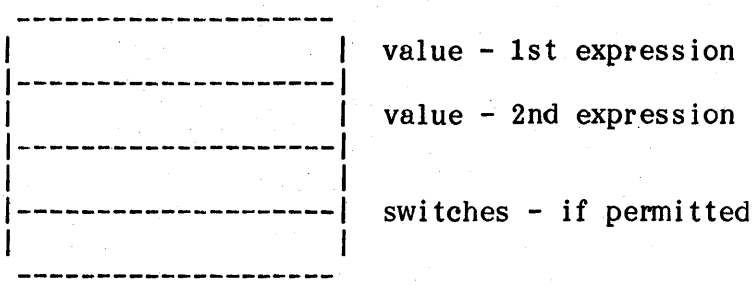
The corresponding field will be expected to conform to the syntax for a file name reference. Seven words are required in the data area for a file name plus two additional words for switches if allowed (see FLAGS, Bit 13) for a total of nine words for this field type.



All characters are packed left to right from the origin of their respective sub-fields, with unused character positions containing nulls.

FIELD TYPE, BIT 14 - NUMERIC RANGE

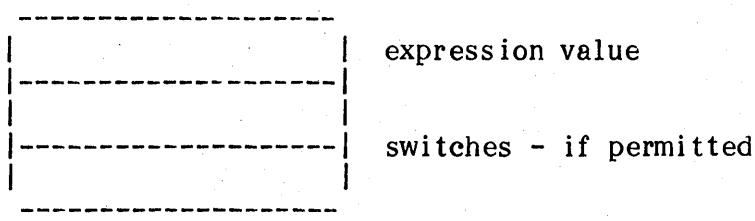
The general syntax for a numeric range will be used to interpret the corresponding field. The default mode for numeric conversion is decimal unless overridden in the field itself, or via FLAGS, Bit 14 in the table entry. Two words are required for the range values and two for switches (if permitted).



The syntax for a numeric range allows a single expression to suffice. In such cases, both value entries in the data area will contain the value of the single expression.

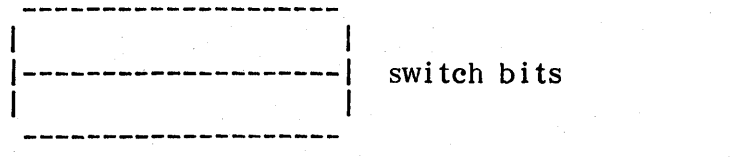
FIELD TYPE, BIT 13 - NUMERIC EXPRESSION

The data area for this field type requires only a single location for the expression value plus two words for switches (if permitted).



FIELD TYPE, BIT 12 - SWITCHES

The syntax for a "switches only" field is exactly the same as switches appended to another field. The data area for this field type requires only two words for the switches.

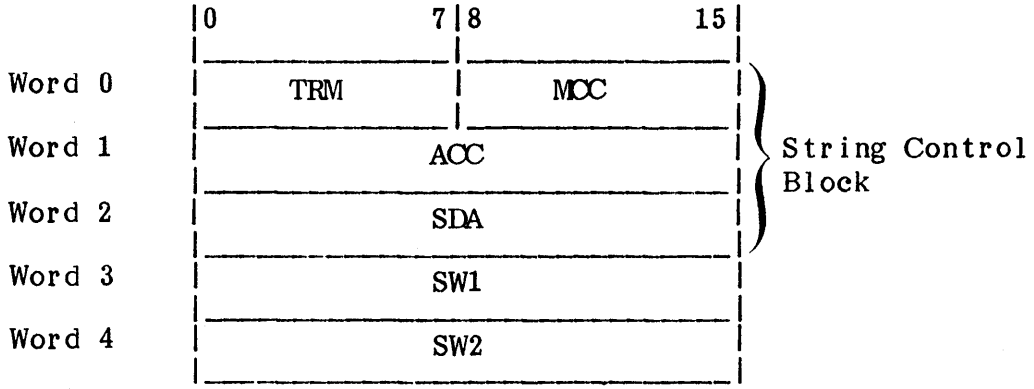


FIELD TYPE, BIT 11 - FILE NAME REFERENCE PATTERN ALLOWED

The syntax and data area for this field is the same as a file name reference field with the exception that the characters "*" and "-" are allowed as pattern specifiers.

FIELD TYPE, BIT 10 - STRING FIELD

Upon encountering a string type field, the parser program will begin moving characters from the input command line to the specified data area. Movement will continue until end-of-line is encountered or until either the specified character count is exhausted or the specified terminating character is encountered. Associated with this field type is a three-word control block, addressed by word DATLOC of the field entry. The control block is of the form:



where:

TRM — String terminator. Contains the string termination character. The parser program will move characters from the input command line to the associated data area until this character is encountered or end-of-line is reached. The terminating character will not be transferred to the data area. If the value specified is zero, the data movement will terminate when either a space or a comma character is encountered, or when end-of-line is reached.

MCC — Maximum character count. The value of MCC is equal to the maximum number of characters the user can accept into the data area. The value of MCC cannot exceed D'255. If this count is exhausted and the next character is not equal to the terminating character, the parser program will return control to the user at the error return with error code ERSLE specified. A count equal to zero will cause the user to regain control at the error return with error code ERSLE specified. The maximum character count does not include the string terminator.

ACC — Actual character count. The parser program will place in this location a value equal to the number of characters transferred to the data area. If error action has been taken with respect to this string field, ACC will contain an invalid value.

SDA — String data area address. The first word address of several locations in memory where data associated with this field are to be placed. Characters are transferred from the input command line and packed into the data area two per word. The data area must be large enough to contain the maximum allowable number of characters (MCC). The data area is not cleared prior to moving in the current string, but a null (X'00) is placed after the string if there is room.

SW1/2 — Two words reserved for switches, if permitted. If switches are disallowed, these two words need not be reserved.

FLAGS

Four bits are currently assigned meanings in the table entry flag word:

FLAG, BIT 15 - FIELD "USE" FLAG

This bit in every table entry is cleared when the parser program is invoked. Then, as the command line is parsed, the "use" flag is set in each table entry that is used (in the sense that a field was found to correspond to the entry).

FLAG, BIT 14 - HEX NUMERIC MODE FLAG

The default conversion mode assumed for numeric fields is decimal. This bit, when set, changes the default conversion mode to hexadecimal. Note as well that the command syntax allows the mode of a numeric field to be specified by prefixing either an "X" or "D" to the digit string. These imply either hexadecimal or decimal, respectively. This hit is ignored if the field is not numeric.

FLAG, BIT 13 - SWITCHES FLAG

The general syntax permits switches to be appended to any field type. If switches are permitted for a particular field, the corresponding table entry should have this bit set. Note that switches require two additional words in the data area. Use of this bit requires allocation of switch space in the corresponding data area.

FLAG, BIT 12 - MANDATORY FIELD

If this bit is set in the flags word of a particular field entry, there must exist in the command string a field corresponding to the field entry. The parser program, upon successful completion, checks all the field entries for flag bit 12 on; if at least one field entry has its field use bit (15) off and its mandatory field bit (12) on, the error return will be taken with error code ERMFNU.

FLAG, BIT 11 - FILE NAME-FIELD PATTERN ENCOUNTERED

This bit is set when a pattern has been detected while processing a file name field which allows a pattern to be specified. This bit is cleared in every file name type table entry when the parser program is invoked.

SWITCHES

The general command line syntax permits switches to be associated with any field. Switches are of the form "/alpha" and may be strung together to arbitrary lengths.

When permitted (see FLAGS, Bit 13) and when encountered in the command line, these switches will be encoded and stored into two words included as part of the field's data area.

Since switches are restricted to single letter alphabetic characters, there are only 26 possibilities. The appearance of a letter as a switch causes a bit to be set in the switch words of the data area.

The correspondence is:

Letter	Word	Bit
A	0	0
B	0	1
C	0	2
D	0	3
.		.
.		.
.		.
P	0	15
Q	1	0
R	1	1
.		.
.		.
.		.
Z	1	9

The corresponding bit will be set to show that a letter appeared. The switch words will be initialized to zero before switch interpretation begins.

PARSE COMMAND LINE (CMDPAR)

```
CALL
.WORD CMDPAR
error return
normal return
```

Entry:	AC3 = Address of syntax table.
Error Exit:	AC0 = System error code.
	AC1-AC3 = Unchanged.
	Syntax table changed as noted below. Each token value set into its buffer.
Normal Exit:	AC0-AC3 = Unchanged.
	Syntax table changed as noted below. Each token value set into its buffer.
Error Codes:	ERCMD Command error. ERMFNU Mandatory field not used. ERSLE String length error.

CMDPAR parses the specified command line into the specified fields.

Parsing stops when each of the fields specified by the syntax table has been found, or when a line terminator is encountered. A line terminator is any byte less than X'20 (except TAB, X'09) or the exclamation mark (!). Because the exclamation mark is treated as a line terminator, a comment beginning with an exclamation mark may be appended to a command line.

The order in which non-keyed, non-switches-only fields occur in the command line must be the same order in which their field specification blocks occur in the syntax table. When a non-keyed, switches-only field is encountered in the command line, the first unused switches-only field specification block is used. Any keyed field may appear anywhere in the command line.

Each time a field is found, the use flag bit for the field is set in the syntax table. Each time that an asterisk or hyphen is found in a pattern field, the PTRN (X'0010) flag bit for the field is set in the syntax table.

A non-string field buffer is zeroed only when the field is encountered. A string buffer is never zeroed, but a null byte (X'00) is stored following it if there is room. Hence, a default value for an optional token can be set into a token buffer before calling CMDPAR.

CMDPAR returns error code ERCMD if an illegal character is encountered, if a string token is longer than the maximum length specified for it, if a token not allowed by the specified syntax table is encountered, or if the unkeyed token order is not allowed by the specified syntax table. CMDPAR returns error code ERMFNU if a mandatory token is not encountered. If a string exceeds the maximum length specified for it, CMDPAR returns error code ERSLE.

GLOBAL CONSTANTS

A collection of global constants may be used with programs that call the Command Line Parser. They are defined in the module CPLNK.RB and are listed below:

Syntax Table Header

Value	Global Symbol	Meaning
0	LINLOC	Pointer to Command Line
1	CHRPOS	Current Character Scan Position
2	FLD01	Offset to First Field Packet

Offsets Into Field Packets

Value	Global Symbol	Meaning
0	DATLOC	Pointer to Data Block
1	FLDID	Field Type Entry
2	FLAGS	Field Flags Entry
3	FLDSIZ	Field Packet Size

Field Types

Value	Global Symbol	Meaning
1	FNAM	File Name
2	NRNG	Numeric Range
4	NEXP	Numeric Expression
8	SWCH	Switches Only
16	PTRN	File Name - Pattern Allowed
32	STNG	String Field

Flag Bits

Value	Global Symbol	Meaning
1	USED	In-Use Bit
2	HEXI	Hexadecimal Flag Bit
0	DECI	Decimal Flag = HEXI Off
4	SWON	Switches Included Bit
0	SWOF	No Switches
8	MANDF	Mandatory Field Bit
16	PTRN	File Name Pattern Encountered

GENERAL SYSTEM II COMMAND LINE SYNTAX

Notation key:

::=	is generated from
|	inclusive or
<x>	element of type "x"

<line>	::=	<comment marker> anything|<statement>
<comment marker>	::=	* | !
<statement>	::=	<field>|<statement> <delimiter> <field>
<delimiter>	::=	comma|blank(s)|comma followed by blank(s)
<field>	::=	<argument>|<argument> <switches>
<argument>	::=	<alpha> = <parameter>|<parameter>
<parameter>	::=	<file>|<expr>|<range>|<switches>
<range>	::=	<expr> : <expr>|<expr>
<expr>	::=	<num>|<num> + <numz>|<num> - <num>
<switches>	::=	/ <alpha>|<switches> / <alpha>
<alpha>	::=	any single uppercase letter
<num>	::=	<sign> <dec value>|<sign> <hex value>
<dec value>	::=	<sign> <dec string>|<sign> D' <dec string>
<hex value>	::=	<sign> <hex string>|<sign> X' <hex string>
<dec string>	::=	string of decimal digits (range 0:32767)
<hex string>	::=	string of hex digits (range 0:FFFF)
<sign>	::=	+|-|null
<file>	::=	<device> : <fileref>|$ <fileref>|<fileref>
<fileref>	::=	<filename> . <extension>|<filename>
<device>	::=	<name>
<filename>	::=	<name>

<name>	::=	uppercase letter, followed by any combination of 0 to 5 numerals and uppercase letters
<extension>	::=	any combination of 1 or 2 numerals and uppercase letters

(THIS PAGE INTENTIONALLY BLANK)

COMMENT SHEET

Jacquard Systems

FROM

Name _____

Business Address _____

_____

_____

Does this publication meet your requirements? yes ☐ no ☐

If not, please explain. _____

_____

_____

_____

Do you wish a reply? yes ☐ no ☐

COMMENTS

Describe any errors and/or suggested changes. Please include page number.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

CUT ALONG DOTTED LINE

(FOLD)

FIRST
CLASS
PERMIT
NO. 976
INGLEWOOD,
CA

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN UNITED STATES

POSTAGE WILL BE PAID BY:

Jacquard Systems

P.O. BOX 6044
INGLEWOOD, CA 90312

Attention: Software Support
Documentation Department

(FOLD)

CUT ALONG DOTTED LINE

3340 Ocean Park Boulevard
Santa Monica, California 90405

Jacquard Systems

3340 Ocean Park Boulevard
Santa Monica, California 90405

Jacquard Systems