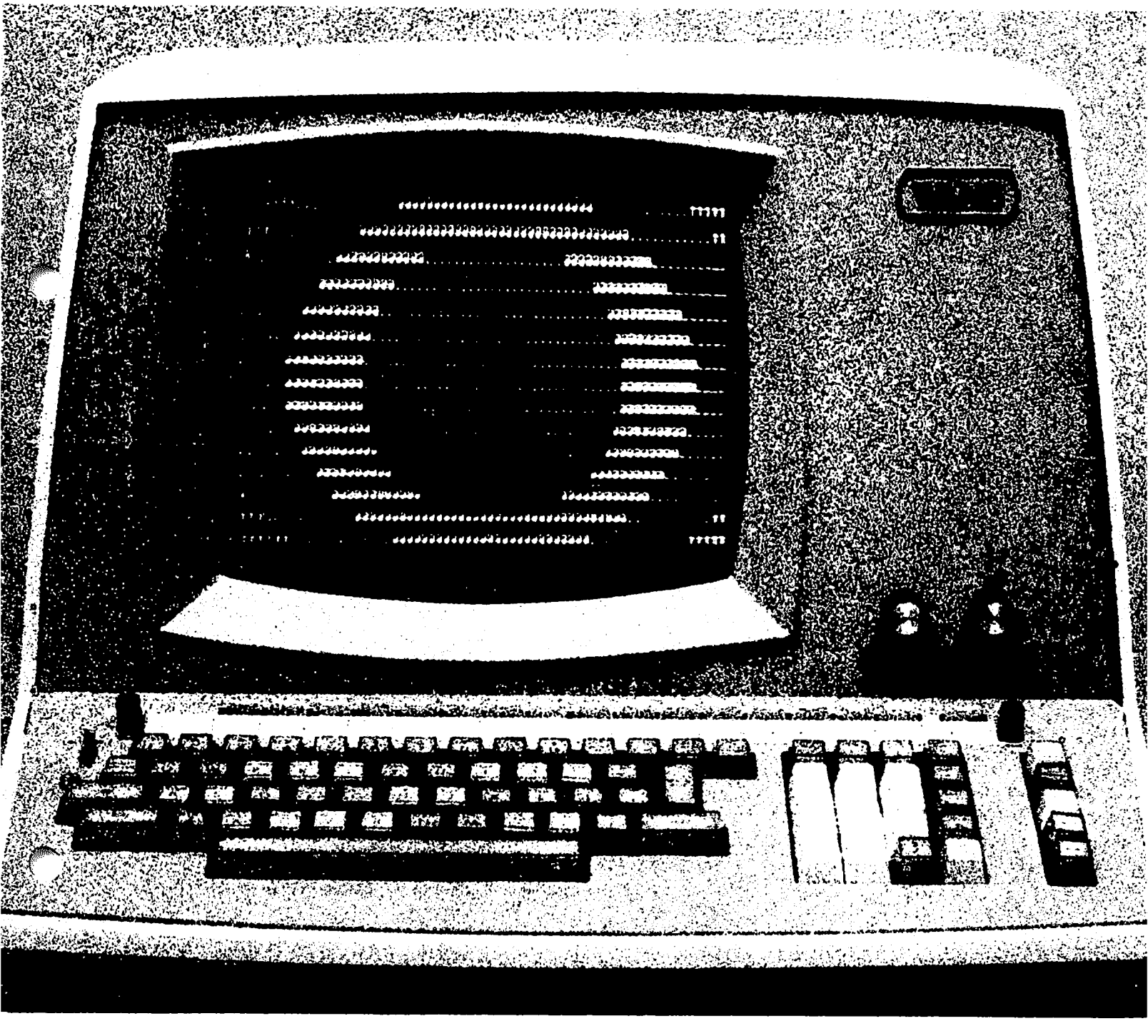
WANG

INTEGRATED SUPPORT SYSTEM USER MANUAL RELEASE 3.7

SYSTEM 2200



INTEGRATED SUPPORT SYSTEM USER MANUAL RELEASE 3.7

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 851-4111, TWX 710 343-6769, TELEX 94-7421

Disclaimer of Warranties and Limitation of Liabilities

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

NOTICE:

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

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 851-4111, TWX 710 343-6769, TELEX 94-7421

HOW TO USE THIS MANUAL

This manual provides a guide for using the collection of software known as the Integrated Support System (ISS) Release 3.7. ISS Release 3.7 is designed exclusively for use on the Wang 2200MVP Central Processor and consists of ISS utility programs, the Key File Access Method (KFAM) Release 7, the ISS Screen/Disk Subroutines, and the SORT-4 Disk Sort Subsystem.

This manual is organized as a reference document for performing such functions as operating a utility program or incorporating supplied subroutines into an application program. Because of its size and the fact that it is a reference manual, the Table of Contents should first be consulted when seeking specific information.

The content of this manual, as revealed by its distinct parts, is organized as follows:

1. Part I of this manual (Chapters 1-4) lists ISS requirements, provides an overview of all ISS support software, furnishes partition generation guidelines, and supplies operating procedures and descriptions necessary to use ISS start-up software.

2. Part II of this manual (Chapters 5-16), supplies operating instructions for each ISS Utility Program. ISS Utility Programs allow files to be copied from disk to disk, provide printed reports of a disk file's contents or the contents of a disk's catalog index, and allow program files to be compressed, decompressed, listed, cross-referenced, or compared to other program files. Other special-purpose utilities are also provided.

3. Part III (Chapters 17-28) describes the Key File Access Method (KFAM), which consists of utility programs and marked subroutines. Utility programs, for instance, allow a KFAM key file to be created, which provides an index (using keys) to records in the parent data file. KFAM global subroutines allow KFAM files to be opened and closed, records added or deleted, and other functions performed. KFAM utility programs create or recreate the key file from the data file, print the key file's contents and the date file's access table, convert KFAM-3 or KFAM-4 files to KFAM-7 format, allow recovery from operational accidents, and reorganize the KFAM file (data file and key file). KFAM files may be accessed simultaneously by multiple memory partitions (multistation).

4. Part IV (Chapters 29-33) describes a variety of marked ISS Screen/Disk subroutines which provide standard functions necessary in a disk-based programming environment. The programmer selects those subroutines required from a displayed list, and upon completion, the subroutines selected are saved into a user-specified disk file as either global or non-global subroutines. Functions provided are associated with operator display/keyboard interaction, disk file access and maintenance, and multistation file access according to one of four access modes. (Multistation files are simultaneously accessible to multiple stations, similar to KFAM files, in several access modes.)

5. Part V (Chapter 34) furnishes programming procedures necessary to use the SORT-4 subsystem, which allows an input file to be sorted according to the ascending or descending order of up to ten sort key fields, input records may be screened before inclusion in the sort, and specification of one of three types of sorts is available.

6. Appendices A, B, and C contain specific information referenced in preceding portions of the manual.

It is assumed that those using this manual are familiar with the operation of all hardware involved as described in other manuals (also see below).

Should a hardware error (ERR lnn form) ever occur, first consult the operating instructions for the ISS software in use. If the error is not listed in the operating instructions error list, consult the error list in the Disk Reference Manual for disk-related errors or the BASIC-2 Language Reference Manual for other errors.

Programmers using ISS or KFAM-7 subroutines or subsystems should have a working knowledge of Automatic File Cataloging statements, as well as an understanding of the Device Table characteristics.

The ISS Release 3.7 package number is 195-0049-3 and consists of four diskettes and a copy of the Integrated Support System (ISS) Release 3.7 User Manual.

ISS 3.7 Diskette numbers and names are as follows:

Diskette Number	Diskette Name
701-2388	ISS UTILITIES
701-2389	KFAM-7
701-2390	ISS SCREEN/DISK SUBROUTINES
701-2391	SORT-4

TABLE OF CONTENTS

NUMERIC LIST OF ISS AND KFAM MARKED (DEFFN') SUBROUTINES

DEFFN'	NAME	TYPE OF SUBROUTINE
200	Data Entry	ISS Screen
201	EBCDIC to ASCII	ISS Trans. Table
202	ASCII to EBCDIC	ISS Trans. Table
203	2200 to 1200	ISS Trans. Table
204	1200 to 2200	ISS Trans. Table
212	Findprevious	KFAM
213	Reopen	KFAM
214	Write Recovery Info.	KFAM
215	Set/Release Hog Mode	ISS Disk
217	Multistation Open	ISS Disk*
218	Multistation End	ISS Disk*
219	Multistation Close	ISS Disk*
220	Enter Gregorian Date	ISS Screen
221	Convert Greg. to Julian	ISS Screen
222	Enter Julian Date	ISS Screen
223	Convert Julian to Greg.	ISS Screen
224	Convert Julian to Proper	ISS Screen
225	Calculate Days Betw. Dates	ISS Screen
226	Limits Next	ISS Disk
227	Free Unused Sectors	ISS Disk
228	Allocate Data File Space	ISS Disk
229	Search Index	ISS Disk
230	Open	KFAM
231	Delete	KFAM
232	Findold	KFAM
233	Findnew	KFAM
234	Findnew (Here)	KFAM
235	Findfirst	KFAM
236	Findlast	KFAM
237	Findnext	KFAM
238	Release	KFAM
239	Close	KFAM
240	Open Output	ISS Disk
241	Close Output	ISS Disk
242	Print	ISS Screen
248	Position Cursor	ISS Screen
250	Open Input	ISS Disk
251	Close Input	ISS Disk
254	Operator Wait	ISS Screen
255	Re-enter	ISS Screen

*Please note that KFAM-7 includes its own version of the Multistation subroutines for non-KFAM files in the global program file KFAM0107. Although the DEFFN' numbers are the same, the subroutines included with KFAM-7 are indeed different than the corresponding ISS Disk subroutines.

PAGE

LIST OF ILLUSTRATIONS

LIST OF TABLES

PART I

OVERVIEW OF ISS

AND START-UP SOFTWARE

OPERATING INSTRUCTIONS

CHAPTER 1
INTRODUCTION

1.1 ISS SOFTWARE COMPONENTS

The Integrated Support System (ISS) provides a Wang 2200MVP system with a combination of utility programs, subroutines, program-controlled routines, and system start-up software which collectively fulfill a wide range of data processing and programming needs.

Utility programs supply aids essential to productive programming, as well as providing standard functions necessary for disk file maintenance and use. Subroutines, chosen from a displayed subroutine list, simplify all programming tasks. There are ISS utility programs and subroutines, and there are the utility programs and subroutines which make up the Key File Access Method (KFAM) Release 7 (usually referred to as KFAM-7).

Sorting disk file records is accomplished by a fast and extremely versatile subsystem called SORT-4. SORT-4 is loaded by a user-written program and thus does not appear on a menu. Another program-controlled routine is included as part of the KFAM system, called Reorganize Subsystem, which like SORT-4 does not appear on a menu.

ISS start-up software is accessible via the module "START", which facilitates updating standard system data, allows menu access to ISS software components and application programs, and makes standard system data available to all software in the 2200MVP memory partition in use.

1.2 HARDWARE AND OPERATING REQUIREMENTS

1. ISS requires dual platter handling capability with at least one diskette drive. All platters other than the four diskettes upon which ISS software is prerecorded must be formatted and scratched using the SCRATCH DISK statement prior to use. A "platter" can be a disk or diskette.

2. ISS Release 3.7 requires a Wang 2200MVP Central Processor.

3. All ISS Utility Programs require a 12K memory partition, with the exception of Program Compare which requires 16K. The KFAM-7 Utility Programs require a 9K partition and one 9K global partition. More detailed information applicable to all ISS software is provided in Chapter 3.

4. A printer is required for all ISS Utility Programs and for the KFAM-7 Utility Programs. Other hard copy output devices may be used, but because top-of-form use is not supported, multipage output is not recommended.

5. Requirements for KFAM-7 are listed in Chapter 18.

6. To make a copy of a disk containing multistation files, use either a COPY statement or the ISS Utility Copy/Verify (input mode = ALL). If KFAM files are copied, see Section 18.5 in this manual. The use of a MOVE statement or a DATASAVE DC END will destroy the multistation file's access table and thus destroy the multistation file. Note that program files cannot be multistation files.

CHAPTER 2
OVERVIEW OF ISS SOFTWARE

2.1 ISS START-UP SOFTWARE

ISS provides station (partition) start-up software in the form of a program module called "START". Station start-up procedures provide access to all ISS support software and to user application programs. Standard data related to the partition in use is made available to ISS support software and application programs.

ISS start-up software is contained on the ISS UTILITIES diskette, on the KFAM-7 diskette, on the ISS SCREEN/DISK SUBROUTINES diskette, but not on the SORT-4 diskette.

2.2 ISS UTILITY PROGRAMS

ISS Utility Programs are operator-controlled routines. Each processing operation performed by ISS Utility Programs (except Disk Dump) can be performed on multiple files. All are compatible with multistation or disk multiplexed files. Their functions are summarized below:

1) COPY/VERIFY - Copies files from one platter to another and verifies the copy. Media conversion may occur during platter-to-platter copy by simply specifying the appropriate device addresses of the drives to be used. Additional sectors may be added to the copied files. Copied files may be renamed, or may replace existing files on the output platter. Files to be copied may be specified directly during Copy/Verify operation, indirectly by means of a reference file, or by means of alphabetical file name limits. Also, all files may be copied from a specified platter.

2) CREATE REFERENCE FILE - Creates a reference file which contains pairs of file name entries for indirect use by the Copy/Verify or Program Compare Utility Programs.

3) SORT DISK CATALOG - Prints a disk catalog index report, with files sorted (1) alphabetically by file name, (2) numerically by starting sector address, or (3) by file sequence in the index.

4) DISK DUMP - Prints the hexadecimal code and graphic character equivalents of the contents of any one disk file. In addition, a data file's contents may be printed with a field-by-field description.

5) DECOMPRESS - Copies a program file and in doing so breaks up all multi-statement lines, assigning a unique line number to each BASIC statement. Files may be specified by file name or by alphabetical file name limits. Also, all program files on a disk may be decompressed.

6) LIST/CROSS-REFERENCE - Prints a list of a program file with each BASIC statement printed on a separate line. For each input program file, it prints four cross-reference tables: one which associates referenced line numbers with the lines which refer to them, one which associates all variables with the lines in which they appear, one that identifies the locations of all marked (DEFFN') subroutines, and one which associates all DEFFN' subroutines with the lines which refer to them. Files may be specified by file name or by alphabetic file name limits. Also, all program files on a platter may be listed/cross-referenced.

7) COMPRESS - Reduces the size of source program files by eliminating REM lines, extra spaces, and inessential line numbers. Files may be specified by file name or by alphabetic file name limits. Also, all program files on a platter may be compressed.

8) RECONSTRUCT INDEX - Reconstructs a disk catalog index in the event of its accidental destruction.

9) FILE STATUS REPORT - Performs several functions tailored to a multistation disk environment, including closing one or all files open to a station, printing the station status of one or all files, and printing all files currently open to a station.

10) PROGRAM COMPARE - Compares two program files on a line-by-line basis, and indicates statements that do not match, if a statement number exists in one program but not in another, if one program ends before another, and if they end at the same statement (statement numbers, file names, and device addresses are listed). The pairs of program files to be compared reside on different platters and may be specified directly by file name, indirectly by a reference file, by alphabetic file name limits, or all program files. With the latter two, files of the same name are compared.

11) FORMAT 2260C DISK - Formats the specified fixed/removable disk in a Model 2260C Disk Drive.

2.3 KEY FILE ACCESS METHOD (KFAM-7)

KFAM Release 7 (KFAM-7) is a software system designed to efficiently produce, search, and maintain an index to the records in a disk-based data file. The index (Key File) is kept as a cataloged file on disk. KFAM-7 includes global subroutines which are accessible to all KFAM utility programs and user written application programs. These marked subroutines perform all the routine operations on the index: random access search, sequential access search, adding and deleting entries.

Because KFAM-7 maintains an indexed entry for each record in the data file, records may be added in random order of their keys. Supplied utility software reorders the data file records and their corresponding index entries into ascending key sequence order for efficient ascending or descending key sequence access using KFAM-7 subroutines. KFAM-7 also includes utility programs that set-up a new KFAM-7 index and carry out recovery and occasional maintenance tasks on a file.

KFAM-7 is fully compatible with multistation disk file operation and controls file access for all stations with control information in global memory. KFAM-7 does not support multiplexed disk file access.

2.4 ISS SCREEN/DISK SUBROUTINES

There are three groups of marked (DEFFN' statement) subroutines which collectively are known as the ISS Screen/Disk subroutines. They are the Screen, the Disk, and the Translation Table subroutines. The subroutines desired from any of the three groups may be selected and saved into a disk file for subsequent loading. The subroutines may be specified as either non-global subroutines, which are incorporated into a user's application program, or global subroutines where only certain variables are incorporated into the user's application program. The actual global subroutines are loaded and run in a background (non-interactive) partition and are accessible to all partitions in the 2200MVP system.

1) Search Index: This Disk subroutine examines the Disk Catalog Index to determine if a particular file: (1) has been cataloged, (2) is scratched or active, and (3) is a data or program file. The function performed is similar to the LIMITS statement.

2) Allocate Data File Space: This Disk subroutine opens a data file on any selected disk, and allocates to it the available sectors between the current end and the end of the cataloged area. It checks the index to ensure the uniqueness of the file name and allows a minimum acceptable file size to be specified.

3) Free Unused Sectors: This Disk subroutine examines a selected file in a catalog area, de-allocates those sectors between the DATASAVE DC END trailer and the end of the file, and repositions the end of file control sector. The de-allocation may be restricted by specifying that a minimum number of extra (reserved) sectors be maintained in the file area.

4) Data Entry: This Screen subroutine accepts a numeric or alphanumeric keyboard entry, using the LINPUT statement, and checks a numeric entry to ascertain whether it is within a specified range and whether its length, and number of places before and after the decimal, is acceptable. It also displays a prompt and optionally an operator-modifiable default value.

5) Open/Close Output: These Disk subroutines open for output or close data files containing certain special purpose, software header and trailer records.

6) Open/Close Input: These Disk subroutines open for input, or close, data files containing certain, special purpose, software header and trailer records.

7) Position Cursor: This Screen subroutine moves the cursor to any point on the CRT and, optionally, erases characters to the right of the new cursor position and the lines below it. The functions available are similar to the PRINT AT function. Usually a PRINT or LINPUT statement follows cursor positioning.

8) Date: This group of Screen subroutines converts and manipulates dates in Gregorian and Julian form. It includes a subroutine for operator entry of the date.

9) Operator Wait: This Screen subroutine displays the message "KEY RETURN(EXEC) TO RESUME" and waits on an INPUT instruction for depression of RETURN(EXEC).

10) Limits Next: This Disk subroutine returns the name of the next file, in the order of file entries in the disk index, and indicates whether the file is a program or data file and either scratched or active.

11) Multistation Open/End/Close: These Disk subroutines provide a controlled file access system for data files on a multistation disk drive. This access system is made possible by maintaining file access information in the file's (hardware-generated) catalog trailer record. Four access modes are available including Exclusive file access. Multistation file Open/End/Close subroutines support creation of a new file, accessing an existing file, changing access modes without closing a file, writing an END record, closing a file, and file password protection. Multistation Disk subroutines are compatible with multiplexed disk file access. A disk hog mode option using the $OPEN and $CLOSE statements is available.

12) Print: This Screen subroutine allows a specified character to be printed a specified number of times.

13) Reenter: This (internal) Screen subroutine displays REENTER to indicate invalid operator entries.

The Translation Table subroutines set up a table (an alphanumeric array) for use with the BASIC-2 statement $TRAN. Four subroutines are provided which assign the proper hex codes for the following translations:

EBCDIC	TO	ASCII
ASCII	TO	EBCDIC
2200	TO	1200
1200	TO	2200

2.5 SORT-4 SUBSYSTEM

SORT-4 is a program-controlled routine for sorting records in a disk file. A user's set-up program provides parameters for the sort operation and loads the SORT-4 module. Up to ten ascending or descending sort key fields determine output record order. SORT-4 provides fast and versatile sort operations including tag sorts, key sorts and full-record sorts, a variety of acceptable input file and input record formats, input record selectivity features, and other options.

CHAPTER 3
ISS START-UP INSTRUCTIONS AND RELATED INFORMATION

3.1 INTRODUCTION

ISS start-up operation allows the user to view and optionally modify certain information pertaining to the 2200MVP memory partition currently in use. Following ISS start-up operation, the information pertaining to that memory partition, including available peripheral addresses and the date, is made available to all software running in that memory partition (via common variables). For instance, during the operation of an ISS utility program, the ISS start-up disk addresses are used to determine if an operator-entered disk address is valid, and the date appears on most utility program printouts.

ISS start-up operation typically occurs only at the start of the working day, that is, after a 2200MVP partition configuration has been executed. During the working day, however, if a 2200MVP partition configuration is re-executed or if the user needs to change a peripheral address, ISS start-up operation must occur again.

ISS start-up information pertaining to a partition is called a system configuration table and consists of certain values obtained either through testing performed internally by ISS during start-up operation or by operator entries in reply to ISS start-up prompts. Information in the system configuration table obtained from operator entries may be stored (recorded) whenever desired during ISS start-up operation on the ISS disk(ette) within a "station file." The values currently stored within a "station file" are recalled (loaded) and used as default values for operator entries during subsequent ISS start-up operations. For example, the default values stored from a previous day's start-up are usually acceptable with the exception of the current date.

A station file may be created during ISS start-up operation for a particular STATION NUMBER and occupies 10 disk "sectors." Because ISS-3.7 operates only on the 2200MVP, it is recommended that the STATION NUMBER be the partition number currently in use. The first prompt which appears during ISS start-up operation requests entry of the STATION NUMBER. Based on the STATION NUMBER entered, the corresponding station file is recalled to obtain default values for ISS start-up operation. In addition, the STATION NUMBER entered is equated to a common variable (scalar variable S2) which is used by ISS utility programs running in that partition and also may be used by application programs. Because the STATION NUMBER is used to determine which user is accessing a multistation disk file, problems may result if the station numbers in use at any time are not unique.

> NOTE:
>
> Although ISS start-up operation allows station numbers from 1 through 48, the SORT-4 Subsystem and ISS utility programs are operable only with station numbers 1 through 4, and KFAM-7 utility programs and subroutines are operable only with station numbers 1 through 16.

3.2 PARTITION GENERATION CONSIDERATIONS

As discussed in the 2200MVP Introductory Manual, the 2200MVP system disk contains the 2200MVP Operating System as well as other software associated with partition generation. During partition generation, available 2200MVP memory is allocated to the partitions to be used, and each terminal may be assigned to one or more partitions. A program file residing on the 2200MVP system disk may be automatically loaded into the partition specified and run if the automatic bootstrap feature has been implemented. Programming may be enabled or disabled for any partition desired. Prior to its execution, a partition configuration may be stored (recorded) on the 2200MVP system disk for future recall and use.

When assigning partition numbers during partition generation, it is recommended that the user assign partitions 1 through 4 to individual terminals which are most likely to use ISS utility programs. In addition, partitions within which interactive programming will occur or operator-interactive software will be run (i.e., foreground programs) should be assigned a lower partition number than application programs requiring little or no operator interaction (i.e., background programs). In general, when assigning partition numbers, the user should keep in mind the recommended partition number/station number convention and plan what software will run in which partition according to station number limitations.

Table 3-1 lists each ISS-3.7 software component and indicates the minimum partition memory size and considerations necessary to run that software component.

Table 3-1. ISS Software Components and Partition Considerations

ISS SOFTWARE	PARTITION SIZE AND CONSIDERATIONS
ISS Start-up Operation	The partition size must be at least 6.75K.
ISS Utility Programs	All require 12K memory with the exception of PROGRAM COMPARE which requires 16K memory. A global partition is not needed.
KFAM-7 Utility Programs	All require 9.0K memory. A 9.0K global partition is required to contain the supplied global program file KFAM0107 (or customized equivalent; see Chapter 18).
KFAM-7 Subroutines	The same KFAM0107 global partition is needed as in the KFAM-7 utility programs. About 1000 bytes plus 87 bytes per KFAM file accessed is needed for KFAM-7 variables in addition to the memory required for the user-written application program statements.
ISS Screen/Disk Subroutines	ISS Screen/Disk Subroutines may be specified as global or non-global. If <u>all</u> ISS Screen/Disk subroutines are chosen for <u>non-global</u> (local) use, a 10.0K partition is needed while saving them to a disk file, and they require 8.25K when RUN, in addition to the memory required for the user's program text. However, it is not likely that all Screen/Disk subroutines will be required, therefore the actual partition size needed will likely be less than 10.0K or 8.25K respectively. If <u>all</u> ISS Screen/Disk subroutines are chosen for <u>global</u> use, a 10.0K partition is required while saving them into two disk files (one file contains the global subroutines, the other contains variables to be added to the user's program). The global subroutines require an 8.25K partition when RUN. The variables require 2.0K when RUN, in addition to the memory required for the user's program text. However, it is not likely that all Screen/Disk subroutines will be required, therefore the actual partition size needed will likely be less than 10.0K, 8.25K, or 2.0K respectively.
SORT-4	If a sequential file is to be sorted, at least a 9K partition is required; if a KFAM file is to be sorted, at least an 11 to 12K partition size is required, depending on the size of the user's set-up program. Also, keep in mind that SORT-4 adjusts itself to the amount of memory available and will provide better throughput if RUN in a larger size partition.

3.3 ISS START-UP PROCEDURES

Prior to ISS start-up operation, the following must have occurred as described in the 2200MVP Introductory Manual.

1. The 2200MVP must be Master Initialized.

2. If the peripheral addresses have not yet been defined in the 2200MVP Master Device Table, choose the "Edit Device Table" option following Master Initialization. This option allows entry of device addresses into the Master Device Table which should subsequently be saved as the default Master Device Table values on the 2200MVP system disk by saving a partition configuration ("Save Partition Configuration" option) before a partition configuration is executed. Once entered into the Master Device Table, the 2200MVP Operating System allows the peripherals specified to be used from the terminals specified. The "Edit Device Table" option is only necessary following Master Initialization if the default Master Device Table values are not acceptable.

3. Create and execute a partition configuration. The automatic bootstrap feature cannot be implemented until the program files (such as KFAM0107) to be automatically bootstrapped (loaded into a partition and run) have been copied to the 2200MVP system disk. To copy program files to the 2200MVP system disk, use Copy/Verify, an ISS utility program, after completing start-up procedures.

4. After executing a partition configuration, "READY (BASIC-2) PARTITION nn" appears in the upper-left corner of the terminal's screen, where nn is the partition number assigned during partition generation. If the terminal in use has been assigned more than one partition, the $RELEASE TERMINAL statement may be necessary to change the partition number currently attached to this terminal. (Refer to the BASIC-2 Language Reference Manual.)

5. ISS start-up operation may now begin.

Please note that the top row of keys on the keyboard are referred to as Special Function Keys. When the SHIFT key is not depressed (the SHIFT light is not illuminated), Special Function Keys '0 through '15 are available; when the SHIFT key is depressed (the SHIFT light is illuminated), Special Function Keys '16 through '31 are available. If the operating instructions below request the operator to touch any Special Function (S.F.) Key, the cursor must not be blinking on the terminal used. A blinking cursor indicates that Edit mode is active. Touching the Edit key once will switch Edit mode off if it was active and replace the blinking cursor with a steady (non-blinking) cursor. A steady cursor indicates that the operator may touch the requested S.F. Key. (Also, refer to the 2200MVP Introductory Manual for related information.)

In the instructions which follow, the word "enter" indicates touching the appropriate keyboard characters, visually verifying the displayed entries, possibly touching the BACK SPACE key to correct a character, and, when correct, touching the RETURN key to indicate completion.

Loading ISS Start-up Software

ISS start-up operation requires the program file START to be loaded as described below following execution of a partition configuration. Once ISS start-up operation has occurred, it is usually not necessary to reload ISS software unless a partition configuration is re-executed.

To load the START program file, complete the following instructions:

1. If an ISS diskette is to be used, the ISS diskette chosen should be the ISS diskette containing the ISS software to be used, and the Write Protect tab must be in place. ISS start-up software is contained on all ISS diskettes with the exception of the SORT-4 diskette. Mount the ISS disk(ette) preferably at disk device address 310.

2. Touch the CLEAR key, and then touch the RETURN key.

3. If the ISS disk(ette) is mounted at disk address 310, complete the procedures which follow. Otherwise, skip to step 4.

    a. Touch the LOAD key, touch the RUN key, and then touch the RETURN key.

    b. Refer to "ISS Start-up Operating Instructions" below.

4. Enter the following commands (terminate each line by touching the RETURN key):

    ```
    SELECT DISK xyy
    LOAD DC T "START"
    ```

    (where xyy indicates the disk device address where the ISS disk(ette) has been mounted.)

5. Refer to "ISS Start-up Operating Instructions" below.

ISS Start-up Operating Instructions

After loading ISS start-up software, the following prompt appears in the upper-left corner of the screen:

```
ENTER STATION NUMBER
?_
```

This prompt requests entry of a station number. Also, it (1) allows a station file to be created for a new station number and (2) allows the operator to view existing station files and the default ISS start-up values contained within each station file. Note that a station file for station number 1 is contained on each ISS disk(ette) containing START.

In reply to the ENTER STATION NUMBER prompt, the following options are available:

a. To view existing station files on the ISS disk(ette) in use and the defaults contained within each, touch S.F. Key '00. "SEARCHING FOR ACTIVE STATIONS" briefly appears, followed by the prompt "ENTER STATION NUMBER TO REVIEW DEFAULTS (0=END)." The EXISTING STATIONS (numbers 01-48) are also displayed. Enter the station number (01-48) whose defaults are to be viewed. Repeat as desired. After viewing the defaults for the appropriate stations, enter 0 (zero); the "ENTER STATION NUMBER" prompt reappears.

   The defaults displayed include the DATE last entered (preset to 05/01/78), the PRINTER ADDRESS (preset to 215), the DISK ADDRESSES (preset to B10), and the LOADING ADDRESS where ISS start-up software was loaded from.

b. To create a station file for a station number on the ISS disk(ette) in use, touch S.F. Key '16. "ENTER STATION NUMBER TO CREATE" appears requesting entry of the station number for which a station file is to be created. Enter the station number; the "ENTER STATION NUMBER" prompt reappears after the new station file has been created.

c. To proceed with ISS start-up operation, enter the STATION NUMBER from 1 through 48, however, it is recommended the station number/partition number convention be adhered to, as described in Section 3.1. If the prompt reappears, this indicates that a station file does not exist on the ISS disk(ette) in use for the station number entered (see paragraph b). Otherwise, the ENTER DESIRED FUNCTION prompt appears (see below).

After a valid station number has been entered, the following prompt appears:

```
ENTER DESIRED FUNCTION (0=END)
?_
    STATION #n
1.  DATE                - mm/dd/yy
2.  PRINTER ADDRESS     - abc
3.  DISK ADDRESS        - xyy
4.  LOADING ADDRESS     - xyy
```

In reply to this prompt, the user views the station number entered (n above) and the default values displayed. If the wrong STATION NUMBER is displayed, touch S.F. Key '31 to re-enter the STATION NUMBER. Otherwise, the user may change any of the displayed defaults desired. When the displayed defaults are acceptable, the user may (1) save the defaults (if changed) into the station file, (2) obtain the Applications menu from which an application program may be loaded, or (3) obtain the menu corresponding to the software contained on the ISS disk(ette) in use.

The user may change one, some, or all of the displayed defaults by completing the following procedures in reply to the "ENTER DESIRED FUNCTION" prompt.

a. To change the DATE, enter 1. The prompt "ENTER TODAY'S DATE" appears requesting entry of the date in the form MM/DD/YY (month/day/year). After entering the date, the "ENTER DESIRED FUNCTION" prompt reappears and the DATE entered is displayed.

b. To change the PRINTER ADDRESS, enter 2. The prompt "ENTER PRINTER ADDRESS" appears and supported printer device addresses are displayed. The printer address entered determines the output device used during ISS utility program operation for certain functions. If a printer is not available, touch RETURN without entering any characters (BLANK or 000). If a printer connected to this terminal is available, enter 204. Otherwise, enter the 3-digit printer address which has been previously set in the Master Device Table. Note that the printer whose address is entered must be ON and SELECTed or else the address is not acceptable (the prompt reappears). After entering the printer address, the "ENTER DESIRED FUNCTION" prompt reappears and the PRINTER ADDRESS entered is displayed.

c. To change the DISK ADDRESSES, enter 3. The prompt "ENTER DISK ADDRESS" appears requesting entry of the disk device addresses to be used by this station, in addition to the ISS LOADING ADDRESS. Supported disk addresses are displayed and any of the displayed disk addresses are accepted with the exception of the ISS LOADING ADDRESS. Please note that ISS start-up operation does not verify (check) that the disk device whose address has been entered is on-line or exists in the Master Device Table. Enter the disk addresses in the form xyy; the entry made is displayed. Enter each disk address desired. Enter 0 (zero) after the last DISK ADDRESS has been entered and displayed; the "ENTER DESIRED FUNCTION" prompt reappears.

d. To change the ISS LOADING ADDRESS, enter 4. The prompt "ENTER ISS LOADING ADDRESS" appears. The ISS LOADING ADDRESS is the address ISS software is to be loaded from, i.e., the disk device address where the ISS disk(ette) is to be mounted. Enter the ISS LOADING ADDRESS; if the disk address entered is one of the displayed DISK ADDRESSES, it is deleted from the DISK ADDRESSES list and displayed as the LOADING ADDRESS. After entering the LOADING ADDRESS, the "ENTER FUNCTION NUMBER" prompt reappears. The ISS disk(ette) should be mounted at its new disk address at this time.

When the displayed defaults are acceptable, the user has the following options available in reply to the "ENTER DESIRED FUNCTION" prompt.

1. To store the currently displayed defaults into the station file, touch S.F. Key '00; the "ENTER DESIRED FUNCTION" prompt reappears. This option allows the displayed defaults to be recalled and used during subsequent ISS start-up operations.

2. To obtain the Application menu, from which an Application program may be loaded, touch S.F. Key '16. Refer to "Loading an Application Program" below for instructions. If the ISS disk(ette) is not mounted at the ISS LOADING ADDRESS, the prompt "ENTER ISS LOADING ADDRESS" appears; refer to paragraph d. above for instructions.

3. To obtain the menu corresponding to the software contained on the ISS disk(ette) in use, enter 0 (zero). The appropriate menu appears unless the ISS disk(ette) is not mounted at the ISS LOADING ADDRESS, which results in the appearance of the "ENTER ISS LOADING ADDRESS" prompt. Refer to paragraph d. above for instructions.

ISS Menu Hierarchy

The ISS menu which appears upon completion of ISS start-up dynamically reflects the ISS software contained on the ISS disk(ette). For instance, if the ISS diskette containing KFAM-7 is in use, the KFAM-7 menu appears and allows selection of a KFAM-7 utility program. Similarly, the ISS UTILITIES menu appears if that ISS diskette is used, or the ISS SCREEN/DISK SUBROUTINES menu appears if that ISS diskette is used. However, if two or more ISS diskettes are copied to a single fixed/removable disk, a higher-level menu called a SYSTEM MENU (see Figure 3-1) appears and allows selection of the software available on the ISS disk. All menus list the options available, which are chosen by touching the S.F. Key corresponding to the desired option. ISS software may be copied to a fixed/removable disk using the instructions provided in Chapter 4.

The SORT-4 subsystem does not appear on any ISS menu because it is not an operator-controlled utility program. SORT-4 is instead controlled by a short, user-written set-up program.

Figure 3-1 shows an example of an ISS SYSTEM MENU, for an ISS disk to which the SCREEN/DISK SUBROUTINES and KFAM-7 have been copied. In reply to the SYSTEM MENU shown, the user would touch S.F. Key '02 to obtain the SCREEN/DISK SUBROUTINES menu or S.F. Key '04 to obtain the KFAM-7 menu. S.F. Key '31 returns the previous menu or prompt (in this case the "ENTER DESIRED FUNCTION" prompt) to the screen.

```
                WANG COMPUTER SYSTEMS (STATION # = 1)
                        SYSTEM MENU

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

  FN KEY    PROGRAM NAME                    FN KEY    PROGRAM NAME

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    02      SCREEN/DISK SUBROUTINES           31      RE-START SYSTEM

    04      KFAM-7
```

Figure 3-1. Example of an ISS System Menu

NOTE:

Following ISS start-up, never remove a disk or diskette containing data or switch the CPU's power OFF unless the ISS Utilities menu, KFAM-7 menu, ISS Screen/Disk Subroutines menu, or the System menu appears on the screen. Similarly, an ISS disk(ette) must not be replaced by another ISS disk(ette) unless the "ENTER STATION NUMBER" prompt appears on the screen, which may be obtained by touching S.F. Key '31 several times.

Loading an Application Program

After correcting the defaults displayed along with the "ENTER DESIRED FUNCTION" prompt, the user may touch S.F. Key '16 to obtain the Applications menu, as shown below:

ENTER OPTION TO CHANGE (0=LOAD APPLICATION)

1. APPLICATION TO LOAD - filename
2. APPLICATION DISK ADDRESS - xyy

(all common variable values are also displayed.)

The default application program file name (filename above) appears to the right of APPLICATION TO LOAD, and the default disk device address from which the application will be loaded (xyy above) appears to the right of APPLICATION DISK ADDRESS. All ISS start-up common variables are displayed (also see Section 3.4).

To change the APPLICATION TO LOAD, enter 1. Enter the file name of the application program to be loaded in eight characters or less. The entry made is displayed and the Applications menu ("ENTER OPTION TO CHANGE" prompt) reappears.

To change the APPLICATION DISK ADDRESS, enter 2. Enter the disk address the application program is to be loaded from. The entry made is displayed and the Applications menu reappears.

If any of the other displayed defaults are incorrect, depress S.F. Key '31 to obtain the "ENTER DESIRED FUNCTION" prompt (see above).

When the displayed defaults are all acceptable, enter 0 (zero) to load the application program specified.

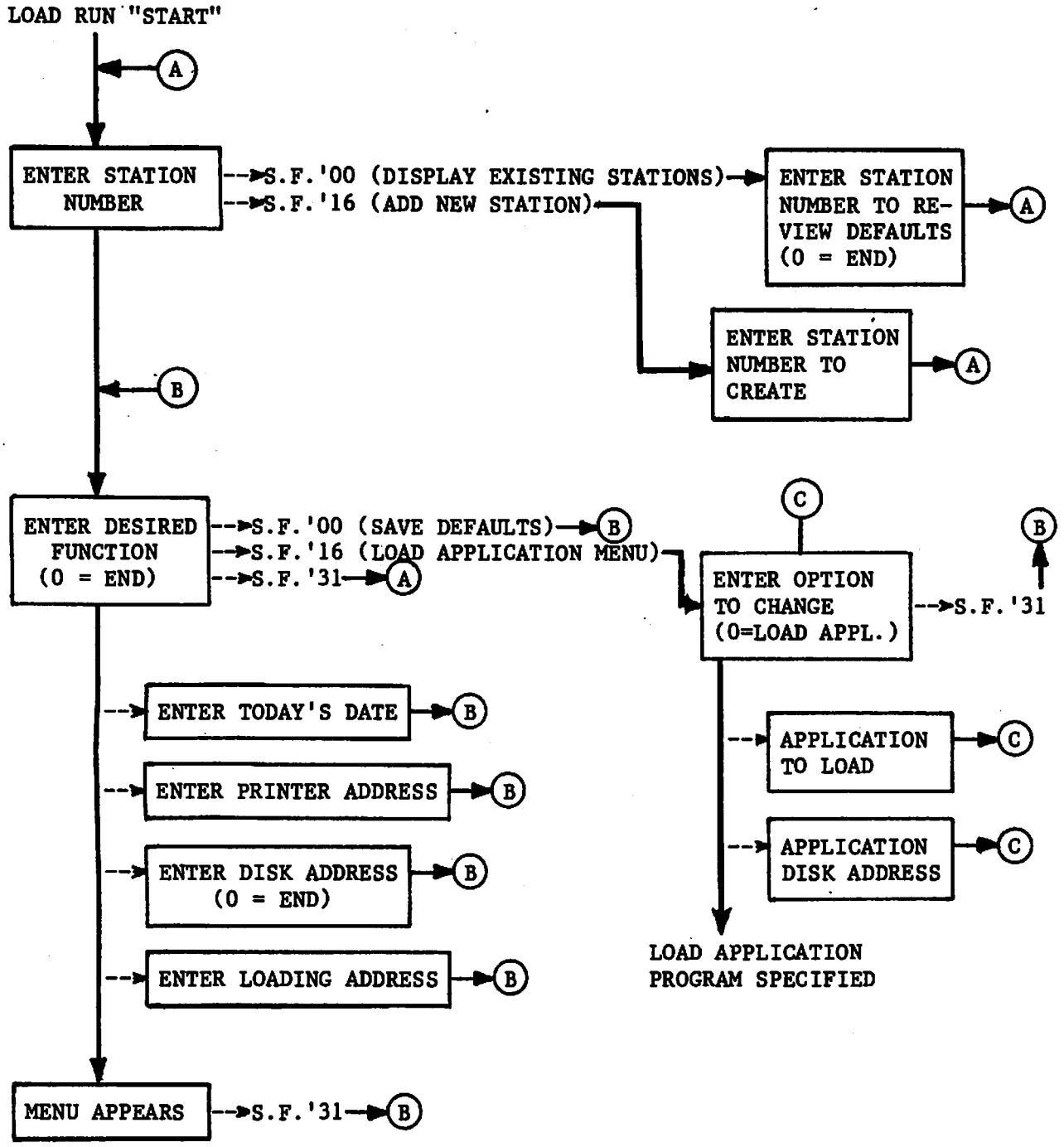
Flowchart of ISS Start-up Operation and Operating Notes

Certain conventions are adhered to by all ISS software. In particular, S.F. Key '31 is available in reply to any prompt during ISS start-up operation to obtain the previous prompt and also allows a user to abort operation of a utility program.

For example, in the flowchart of ISS start-up operation shown in Figure 3-2, touching S.F. Key '31 in reply to an ISS menu (e.g., ISS Utilities menu) returns the "ENTER DESIRED FUNCTION" prompt to the screen (which, for instance, might be needed to change the ISS start-up PRINTER ADDRESS).

It is important to note that S.F. Key '31 is available during the operation of any ISS or KFAM-7 utility program to abort the program instead of entering a reply to a prompt. S.F. Key '31, for instance, should be touched to return the appropriate menu to the screen if the wrong utility program was accidentally chosen (in reply to the ISS Utilities or KFAM-7 menu.) S.F. Key '31 also closes any files left open if a program is aborted.

> NOTE:
>
> S.F. Key '31 is valid only if the terminal in use is not in Edit mode. Edit mode, if active, is indicated by a blinking cursor and may be deactivated by manually touching the EDIT Key once (a steady cursor indicates Edit mode is not active).



Dashed lines indicate options.

Figure 3-2. Flowchart of ISS Start-up Operation

3.4 APPLICATION PROGRAM REQUIREMENTS AND ISS COMMON VARIABLES

Table 3-2 lists the ISS common variables whose values are determined during ISS start-up operation. Following ISS start-up operation, these common variables are available for use in application programs and their values are displayed along with the Application menu.

Table 3-2. ISS Common Variables

DESCRIPTION	VARIABLE NAME
Julian Date	Scalar variable Q1.
Gregorian Date	Alpha-variable Q1$, dimensioned as Q1$8.
Memory Size (K)	Scalar variable S.
CRT Width (64,80)	Scalar variable S0.
Application Loading Number (1=310, 2=320, 3=330, 4=B10, 5=B20, 6=B30, 7=350, 8=360).	Scalar variable S1.
Station Number (1-48)	Scalar variable S2.
CPU Type (4=MVP)	Scalar variable S3.
ISS Loading Address	Alpha-variable S$, dimensioned as S$3.
Printer Address	Array element S$(1), dimensioned as array S$(9)3.
Disk Addresses	Array elements S$(2) through S$(9), dimensioned as array S$(9)3.

Application Program Requirements

Application programs running under ISS should adhere to the following requirements.

1. All numeric and alphanumeric, scalar and array variables whose first letter begins with Q, R, S, T, U, V, or W are reserved exclusively for ISS system use. The variables reserved for SORT-4 are listed in Appendix B.

2. All DEFFN' statements from DEFFN' 200 through DEFFN' 255 are similarly reserved for ISS system use. Also see paragraph 5 below.

3. If an ISS disk(ette) has been removed from the ISS loading address due to application program processing, upon its completion, the application program should furnish a return to ISS by providing the following: (1) issue a SELECT DISK xyy statement, where xyy is the ISS loading address, (2) provide the operator with a means of remounting the ISS disk(ette), e.g., a prompt, and (3) LOAD and RUN the ISS "START" file when the operator signals that the ISS disk(ette) is ready.

4. It is recommended that disks or diskettes containing application programs contain a "START" program file (module) which provides return links to the ISS disk(ette) "START" module via S.F. Key '31 (use a DEFFN' 31 and a RETURN CLEAR statement). Although not required, this is a recommended system convention.

5. Programming conventions applicable to the 2200MVP Operating System should be adhered to. For instance, the current global partition selected is determined via a SELECT @PART statement in the user's application program which allows DEFFN' statement subroutines in the selected global partition to be accessed via GOSUB' statements in the user's application program(s).

CHAPTER 4
COPYING ISS TO/FROM DISK AND DISKETTE

4.1 COPYING THE ISS SYSTEM TO A FIXED/REMOVABLE DISK

Follow the procedures provided below to copy the prerecorded contents of one, two, three, or all four ISS diskettes to a single fixed/removable (hard) disk. The dynamic ISS menu structure will automatically provide a SYSTEM MENU if the fixed/removable disk at the ISS loading address contains the software prerecorded on any two or three of the following ISS diskettes: ISS Utilities Diskette, KFAM-7 Diskette, and the ISS Screen/Disk Subroutines Diskette. Once copied to a single disk, all ISS software may be copied back to individual diskettes as described in Section 4.2.

> NOTE:
>
> Prior to copy, any output disk or diskette platter to receive ISS software should be reformatted and have the SCRATCH DISK statement executed for that platter. This is strongly recommended prior to performing the procedures outlined below and in Section 4.2.

1. It is recommended that the ISS start-up Printer Address be set to 000 (blank) before beginning the procedures that follow. This action is not necessary if (1) the ISS Printer Address is 000 already, or (2) the user does not mind wasting a few sheets of printer paper. To change the ISS Printer Address: load and run START from an ISS diskette located at the ISS loading address, change the Printer Address to 000, and proceed to step 2 below. Error messages (see step 4) will be displayed instead of printed.

2. Mount the ISS Utilities Diskette at the ISS loading address. Obtain the ISS Utilities menu.

3. From the ISS Utilities menu, load the COPY/VERIFY Utility Program by touching the indicated Special Function Key. Set the Copy/Verify parameters as follows (refer to Chapter 6 for operating instructions):

   FUNCTION = COPY/VERIFY
   INPUT ADDRESS = xyy (diskette address)
   INPUT MODE = ALL
   EXTRA SECTORS = 2
   OUTPUT ADDRESS = xyy (fixed/removable disk address)
   OUTPUT MODE = ADD

4. When the prompt "MOUNT PLATTERS AT INDICATED ADDRESSES, KEY RETURN(EXEC) TO RESUME?" appears, mount the ISS diskette to be copied from at the INPUT ADDRESS; mount a recently formatted and scratched disk at the OUTPUT ADDRESS. Begin Copy/Verify execution by touching the RETURN key when ready. During program execution, ignore messages in the form "FILE - filename - CANNOT BE COPIED" if encountered and "FILE - KFAMWORK - DOES NOT VERIFY" if encountered.

5. Upon completion of copy, the ISS Utilities menu reappears. Repeat steps 3 and 4 above for each ISS diskette. After completing Copy/Verify operations for all ISS diskettes, refer to step 6 below.

6. Obtain the ISS Utilities menu and touch S.F. Key '31. In reply to the "ENTER DESIRED FUNCTION" prompt, enter 4 to obtain the prompt "ENTER ISS LOADING ADDRESS". Change the ISS loading address to the disk address of the hard disk. If not already mounted, mount the disk containing the newly copied ISS software at the new ISS loading address.

7. In reply to the "ENTER DESIRED FUNCTION" prompt, touch S.F. Key '00 to save the new defaults.

8. Enter 0 (zero) in reply to the "ENTER DESIRED FUNCTION" prompt; the SYSTEM MENU may appear at this time (see Figure 3-1).

9. If the ISS Printer Address was changed to 000 in step 1 above, repeat the procedures in step 1 to change the ISS Printer Address back to its original value.

4.2 COPYING ISS FROM HARD DISK TO DISKETTE MEDIA

To copy ISS software from a disk to diskette, refer to the steps below.

1. Mount the ISS disk(ette) which contains the ISS Utilities. Obtain the ISS Utilities menu.

2. From the ISS Utilities menu, load the COPY/VERIFY Utility Program by touching the indicated Special Function Key. Set the Copy/Verify parameters as follows (refer to Chapter 6 for operating instructions):

   FUNCTION = COPY/VERIFY
   INPUT ADDRESS = xyy (hard disk address)
   INPUT MODE = INDIRECT
   EXTRA SECTORS = 2
   OUTPUT ADDRESS = xyy (diskette address)
   OUTPUT MODE = ADD

3. When the prompt "ENTER THE NAME OF THE REFERENCE FILE" appears, refer to the table below for (1) the file name to enter, and (2) the corresponding name to be written on the diskette label upon completion of copy (use a felt-tip pen).

4. When the prompt "MOUNT PLATTERS AT INDICATED ADDRESSES. KEY RETURN (EXEC) TO RESUME?" appears, mount the ISS hard disk at the INPUT ADDRESS and a recently formatted and scratched (SCRATCH DISK) platter at the OUTPUT ADDRESS. Begin program execution by keying RETURN when ready.

Output Medium	Reference File Name	Diskette Name
Diskette	ISS.REF1	ISS Utilities
	ISS.REF2	ISS Screen/Disk Subroutines
	ISS.REF4	KFAM-7
	ISS.REF5	SORT-4

5. Upon completion of copy, the ISS Utilities menu reappears. Remove and label the new ISS diskette. Repeat steps 3 and 4 above for each ISS diskette. After completing Copy/Verify operations for all ISS diskettes, refer to step 6 below.

6. Mount a newly copied ISS diskette (at any valid disk address) and reload ISS software from the new ISS diskette. It may be necessary to create a station file(s) and change the ISS Loading Address. See Chapter 3 for instructions.

PART II

THE ISS UTILITY PROGRAMS

CHAPTER 5
INTRODUCTION TO THE ISS UTILITIES

The ISS utility programs provide several standard functions necessary for a disk-based data processing environment. Categories of the functions performed by ISS Utility Programs include Copy Functions, Programming Functions, Catalog Index Functions, Formatting a Disk (Model 2260C only) and Special Purpose Functions.

Table 5-1 shows the categories of ISS utility program functions relative to the basic elements of disk storage. For instance, Catalog Index Functions apply to all files in the Catalog Index. Programming Functions apply only to Program Files, and the Copy Functions apply to both Data Files and Program Files (All Files).

Table 5-1. ISS Utility Programs and Categories

FUNCTIONAL CATEGORY	STORAGE ELEMENT	UTILITY PROGRAM(S)
Copy Functions	All Files	Copy/Verify
Programming Functions	Program Files Program Files Program Files Program Files	Decompress Compress List/Cross-Reference Program Compare
Catalog Index Functions	Catalog Index Catalog Index	Sort Disk Catalog Reconstruct Index
Special Purpose Functions	All Files All Files Data Files	Create Reference File Disk Dump File Status Report
Formatting a Disk	Disk	Format 2260C Disk

All ISS Utility Programs provide displayed messages called "prompts" that require (1) information to be entered, or (2) a manual operation to be performed. Each utility consists of a series of prompts which appear in a given order. The step-by-step instructions which follow for each utility correspond to the prompt-by-prompt display sequence associated with each utility. All ISS Utilities are selected from the ISS Utilities menu by touching the corresponding Special Function (S.F.) Key, after which the selected utility is automatically loaded into memory.

Note that if an ISS utility is chosen from the ISS Utilities menu and the ISS start-up printer address is 000, when attempting to load that program the following may appear:

```
PRINTER REQUIRED FOR THIS PROGRAM
KEY RETURN(EXEC) TO RESUME
```

To remedy this situation, touch the RETURN key and then touch S.F. Key '31 repeatedly as needed to obtain the "ENTER DESIRED FUNCTION" prompt. Change the start-up PRINTER ADDRESS to a valid printer device address if a printer is available (ON and SELECTed). Obtain the ISS Utilities menu and reload the desired program.

A similar message may appear if a printer is not SELECTed. A printer is required for all ISS utility programs.

Because of the consistency of ISS utility program operation, the following general rules apply:

a. For numeric-only entry fields, leading zeros are not required.

b. Disk addresses (i.e., xyz form) are flexible according to how the equipment is configured when installed. Therefore, it is recommended that labels showing disk addresses be placed on each disk drive housing soon after installation. Valid disk addresses are determined during the ISS start-up operation and include the ISS loading address.

c. Unless indicated otherwise, if an invalid entry is keyed, the erroneous field is erased, the cursor is repositioned and REENTER appears below the entry field. With disk addresses, this occurs (1) if the entered address is not a valid ISS address, (2) if the address was already entered (input and output addresses may not be the same), or (3) the default value used by the program is now invalid, in which case the prompt with RE-ENTER will appear when the program is first loaded. When an entry is made outside its limits, valid limits are also displayed. With file name entry, REENTER may indicate that the input file cannot be accessed due to access mode conflicts if a multistation file is accessed.

d. In the prompt-by-prompt instructions which follow, there are two cases where the prompts indicated may not appear. With default values, the prompt which requests whether these values should be saved appears in all cases except, of course, when the previous default values were not changed. REMOUNT ISS PLATTER appears only if the program file (module) required is not on the platter at the ISS loading address.

e. A printer is recommended for multiple-file processing and unattended operation to record error messages. Several programs will load only if the ISS Printer Address is not blank (000) including Disk Dump, List/Cross-Reference, and Program Compare. Execution error messages are discussed under each ISS Utility Program following the Operating Instructions.

f. For printed listings produced by ISS Utility Programs, the page number and ISS start-up date appear in expanded print at the top of each page. The file name, if applicable, also appears at the top of each printed page.

g. With multistation files, ISS utility programs assume that the file will not be scratched between the time the file name is entered until processing has been completed. This includes Copy/Verify and all utility programs in the Programming Functions category; all of which use the Shared file access mode once file processing has begun. The Create Reference File Utility uses Exclusive access. With the Disk Dump Utility, when entering a file name currently being accessed by another station, RE-ENTER will appear if a conflict occurs.

h. Files to be processed by a selected utility program may be specified as follows:

   1) ALL indicates that all files on the selected input disk will be processed.

   2) PART indicates that selected files, but not all files, will be processed. File names are specified during operation of the given ISS utility program, and are subject to on-line checking by the program to ensure, for example, that the input file name just entered is actually on the input disk. Other checks are also performed as explained in each utility program's operating instructions.

   3) INDIRECT indicates that selected files will be processed, and that the file names are specified in a reference file created previously by the Create Reference File Utility. INDIRECT is similar to PART in that selected files are processed. INDIRECT specification eliminates the need for on-line file name entry during operation (not available with all utilities).

4) RANGE indicates that input file names that fall within specified alphabetic limits (upper-lower limits) will be processed. This technique of file name specification allows file names, encoded within certain alphabetic file name limits upon their creation, to be processed without the need to enter the individual file names (not available with all utilities).

i. Should an error occur which interrupts processing, the following operator actions are advised:

1) If a prompt appears or reappears, refer to the appropriate Operating Instructions and (1) reenter the requested information and continue or (2) refer to the next paragraph.

2) To abort the current operation, or if a hardware-related (ERR lnn form) error or one of the error messages shown in Table 5-2 occurs, unless the Operating Instructions for that utility specify otherwise, touch Special Function Key 31. Touch Special Function Key 31 again if the ISS Utilities menu does not appear.

j. For multistation files, a password is assigned to that file upon creation. If the file does not require password protection, a password of blanks should be assigned to that file. Otherwise, a password other than blanks may be assigned.

k. During the processing of a utility, after all parameters have been entered, an execution error message may be printed to the ISS printer address. These execution error messages are listed and explained after the Operating Instructions for each utility, if any such execution error messages exist for that utility program.

In providing step-by-step operating instructions in this section, the following conventions have been adhered to. The verb "enter", used in the INSTRUCTIONS column, as in the sentence "Enter 1 to list only the active items", means that the key specified (1) should be touched followed by the standard terminator key RETURN.

Note that all references to the (EXEC) key indicate the RETURN key.

In contrast to "enter" the verb "key", used in the INSTRUCTIONS column, as in the sentence "Key H to interrupt execution at the end of the current file", means that only the keys specified are to be depressed. If any terminator is required when the verb "key" is used, the terminator is explicit.

The verb "mount", as in the sentence "mount output disk at the specified address", refers to the group of actions required to fully ready the device for its impending function in the system. In general, it is presumed that the operator is familiar with the operation of the hardware components of the system and has access to the manuals which describe them.

The amount of file space cataloged for a file and the file space "used" by a file are established by two control records, each of which occupies a whole sector. The data END record is the sector whose position marks the end-of-live-data in a file and establishes the USED sectors value for a file, and is written by a DATASAVE DC END statement or subroutine equivalent. The catalog trailer (file end) record is the last sector of the file, whose position is established by the DATASAVE DC OPEN statement or subroutine equivalent based on the sectors allocated for this file.

Table 5-2. Open Return Code Error Messages

MESSAGE	CAUSE	RECOVERY
ERROR RETURN CODE FROM OPEN = A	Access mode conflict with other stations prevents the requested file from being opened at this time by this station.	If "KEY RETURN(EXEC) TO RESUME" accompanies this error message, touch the RETURN key; otherwise, touch S.F. Key '31 to obtain the ISS Utilities menu. Retry, or wait and retry. If this error message persists, use the File Status Report Utility to determine if the file was left open accidentally by another station and close the file if it was left open accidentally.
ERROR RETURN CODE FROM OPEN = D	Requested file was not found at the entered address, attempted to open a file already open to this station, or other file disposition conflict. This error could be caused by using station numbers which are not unique (see Section 3.1).	If "KEY RETURN(EXEC) TO RESUME" accompanies this error message, touch the RETURN key; otherwise, touch S.F. Key '31 to obtain the ISS Utilities menu. Retry taking care to correctly enter the file name and platter address. If this error persists, use LIST DC command to determine if the file exists. If the file does not exist, mount the correct platter or recreate the file. If it exists, use File Status Report to determine if this station left the file open accidentally and correct the file disposition conflict.

Table 5-2. Open Return Code Error Messages (Cont'd)

ERROR RETURN CODE FROM OPEN = P	Password conflict for a file requested during this operation.	If "KEY RETURN(EXEC) TO RESUME" accompanies the error message, touch the RETURN key; otherwise, touch S.F. Key '31 to obtain the ISS Utilities menu. Retry the attempted operation.
ERROR RETURN CODE FROM OPEN = S	Insufficient disk space on the specified platter to catalog the file to be created.	If "KEY RETURN(EXEC) TO RESUME" accompanies the error message, touch the RETURN key; otherwise, touch S.F. Key '31 to obtain the ISS Utilities menu. Retry using a different platter, or retry after making more disk space available on a copy of the same platter.

CHAPTER 6
THE COPY/VERIFY UTILITY PROGRAM

6.1 INTRODUCTION

The COPY/VERIFY utility program copies files from one disk platter to another, and verifies the copy. Files are copied up to and including the data END record. Unused sectors are not copied, and additional (reserve) sectors may be added to the output files. Copied files, as specified by the output mode, may be added to the output disk, or they may replace existing files on the output disk. Selected files, all files within alphabetic limits, or all files may be processed, as specified by the input mode. Selected files may be specified directly, during the parameter entry phase, or indirectly, by means of a reference file. If files are specified directly, up to 50 files may be processed. If files are specified indirectly, in a (CREATE REFERENCE FILE) reference file, 999 files may be processed. The copy and verify operations may be executed independently, or sequentially under program control.

With selected file direct copy, input files to be copied/verified are identified by entering input file names. As each file name is entered, the input disk's catalog index is checked to ensure that each input file is indeed an active cataloged file. The same input file name may be entered, thus the same file may be copied as several output files. Entry of an output file name is rejected only if an identical output file name was already entered. Most file name entry errors are thus corrected before processing begins. Extra (reserve) sectors may be individually assigned to each output file with direct copy only.

Output files may replace existing (active or scratched) files, or they may be new files added to the output disk's catalog, or both, during one operation.

Copying is accomplished by read/write operations rather than COPY or MOVE statements. Input files must be cataloged. The "USED" portion of the file is copied, except if used = 1, in which case the entire file is copied.

NOTE:

To copy KFAM-5 files, see Section 18.5 prior to attempting copy.

NOTE:

With multistation disk files, caution should be observed when using Copy/Verify, because it does not check whether another user is accessing that file. For example, while one station is printing a file, another station using Copy/Verify may use that file as an output file in the REPLACE output mode. Also, while a file is being Copy/Verified, it could be accidentally scratched by another user.

6.2 OPERATING INSTRUCTIONS: COPY/VERIFY

DISPLAY	INSTRUCTIONS
1.	1. From the ISS Utilities menu load COPY/VERIFY by touching the indicated Special Function Key.

NOTE:

If the default INPUT ADDRESS or OUTPUT ADDRESS is not a valid ISS disk address, the prompts shown respectively in step 4 or step 7 will appear instead of the prompt in step 2; also, REENTER appears below the prompt. After completing step 4, step 7, or both, go to step 2. Also, all references to the (EXEC) key indicate the RETURN key in the following instructions.

2. ENTER THE NUMBER OF THE DESIRED FUNCTION
?_/

(Functions 1-8 appear with default values for 1-6)

2. To change a parameter, enter its respective number and go to the step listed below. To exchange the input and output addresses, enter 7.

TO CHANGE	ENTER	GO TO STEP
FUNCTION	1	3
INPUT ADDRESS	2	4
INPUT MODE	3	5
EXTRA SECTORS	4	6
OUTPUT ADDRESS	5	7
OUTPUT MODE	6	8

Enter 8 when all parameters are correct. Go to step 9.

3. ENTER THE NUMBER OF THE FUNCTION DESIRED?
?_/

1 - COPY
2 - VERIFY
3 - COPY AND VERIFY

3. If copy only is necessary, enter 1.
If verify only is required, enter 2.
If copy and verify are desired, enter 3.

Go to step 2.

4. ENTER THE INPUT ADDRESS
?---

ADDRESSES AVAILABLE = XYY

4. Enter the device address at which the input disk will be mounted. Valid ISS disk addresses are displayed.
Go to step 2.

5. ENTER THE NUMBER OF THE DESIRED INPUT MODE
?_/

INPUT MODES AVAILABLE

1-ALL	3-RANGE
2-PART	4-INDIRECT

5. Input mode defines what input files will be copied.
Enter 1 to copy all files on the input disk to the output disk.
Enter 2 to copy some but not all files on the input disk.
Enter 3 to copy input files whose names fall within the alphabetical limits.
Enter 4 to copy all files whose names are listed in a reference file on the input disk.

Go to step 2.

6. ENTER THE NUMBER OF EXTRA SECTORS/FILE (-1 = LEAVE AS IS)
?-----/

6. Enter the number of sectors, in addition to those listed as USED in the Catalog Index, to be included in each copied output file. The entered value of extra sectors applies to all output files, with the following exception: With INPUT MODE = PART (step 5), any value entered other than -1 becomes the default value in step 13 and allows each output file to individually be assigned extra sector values.

For all INPUT MODES, the following applies: If -1 is entered, all sectors listed as USED are copied, and the size of each output file is identical to its input file counterpart (UNCHANGED is displayed). Otherwise, any entry up to 65,535 is valid. Sectors beyond the data END record are not copied. If the (Catalog Index) value of USED SECTORS equals 1, this indicates no catalog trailer record, in which all sectors are copied automatically. If VERIFY only was chosen in step 3, this parameter is irrelevant. Go to step 2.

7. ENTER THE OUTPUT ADDRESS
?---

ADDRESSES AVAILABLE = XYY

7. Enter the device address at which the output disk will be mounted. Valid ISS disk addresses are displayed. Go to step 2.

8.	ENTER THE NUMBER OF THE DESIRED OUTPUT MODE ?-/ OUTPUT MODES AVAILABLE 1-ADD 2-REPLACE 3-ADD/REPLACE	8. Enter 1 to choose the ADD mode, where all output files copied have unique file names and are allocated unoccupied disk space. Enter 2 to REPLACE files on the output disk, whose file names will be entered in step 12, with active files copied from the input disk. Enter 3 to ADD output files whose names are unique to the output disk, and also REPLACE files on the output disk with active files with identical output file names. Go to step 2.
9.	DO YOU WISH TO SAVE THESE VALUES AS DEFAULTS (Y/N) ?-	9. To save the currently displayed parameters as the default values for the COPY/VERIFY utility, enter Y. Otherwise, enter N. If INPUT MODE is ALL, (entered in step 5) go to step 18. If input mode is RANGE, go to step 14. Otherwise go to the next step.
10.	MOUNT INPUT PLATTER KEY RETURN(EXEC) TO RESUME?	10. Mount the input disk at the device address displayed. (The ISS disk may be removed if necessary.) When ready, key (EXEC). If the INPUT MODE parameter is: PART, go to step 11. INDIRECT, go to step 15.

11.	ENTER THE NAME OF FILE #N (0=END) ?--------	11. Enter the file name of the Nth input-file to be copied/verified. After entering all input and output file names, enter 0 (zero) and go to step 17. As each file name is entered, COPY/VERIFY checks to ensure the file name is cataloged as an active file on the input disk. If not, the file name entered and NOT ACTIVE FILE appear. Reenter a valid file name. If the file name was previously entered, the file name entered and IS A DUPLICATE INPUT FILE NAME appear with KEY RETURN(EXEC) TO RESUME? To reenter a different input file name, key (EXEC). Go to step 11. To use the file name just entered (i.e., copy one file several times), key X and (EXEC). Go to step 12.
12.	INPUT FILE = FILENAME ENTER THE NAME OF THE OUTPUT FILE (EXEC = SAME AS INPUT) ?--------	12. For the specified input file, enter its corresponding output file name. With the ADD output mode, this is a unique file name on the output disk. With REPLACE output mode, this file name is identical to an existing file's name on the output disk. If EXTRA SECTORS = -1, go to step 11. Otherwise, go to the next step. If the output file name entered is identical to an output file name previously entered, the file name entered and IS A DUPLICATE FILE NAME appear. Reenter a valid file name.

13.	ENTER THE NUMBER OF FREE SECTORS FOR THIS FILE. (DEFAULT =N) ?-/	13. Enter the number of sectors, in addition to those listed as USED in the Catalog Index, to be included in this output file. The default value displayed (N) is the reply entered to step 6. Up to 999 may be entered. Go to step 11; the input and output file names just entered are displayed above the step 11 prompt.
14.	ENTER LOWER LIMIT OF FILE NAMES ?--------	14. Enter the lower alphabetical limit of the file names to be copied/verified. The lower limit and upper limit entries need not be file names, but the lower limit must be alphabetically less than the upper limit, else INVALID RANGE and REENTER appears (in step 15). File names entered as limits are included in copy/verify.
15.	ENTER UPPER LIMIT OF FILE NAMES ?--	15. Enter the upper alphabetical limit of the file names to be copied/verified. Go to step 17.
16.	ENTER THE NAME OF THE REFERENCE FILE ?--	16. Enter the name of the COPY/VERIFY reference file to be used to specify the files to be copied/ verified. (INDIRECT input mode). NOTE: The reference file must presently reside on the input disk. If not, ERROR RETURN CODE FROM 'OPEN'=D, KEY RETURN(EXEC) TO RESUME? appears. Key (EXEC) and reenter the reference file name.
17.	REMOUNT ISS PLATTER KEY RETURN(EXEC) TO RESUME?	17. If removed, this prompt appears requesting that the ISS disk be mounted at the ISS loading address. When ready, key (EXEC).

18.	MOUNT PLATTERS AT THE INDICATED ADDRESSES. KEY RETURN(EXEC) TO RESUME?	18. Mount the input and output disks at the displayed addresses. (The ISS disk may be removed.) When both disks are ready, key (EXEC).
19.	VERIFYING COPYING FILE NUMBER XXX INPUT FILE NAME = FILENAME	19. The processing display indicates the file currently being copied/verified. With the RANGE input mode, SCANNING INDEX FOR FILE NAME first appears. After all files have been copied/verified, the Utilities menu will appear unless the ISS disk was removed in step 18. If removed, MOUNT ISS PLATTER, KEY RETURN(EXEC) TO RESUME appears. Mount the ISS disk at the ISS loading address. When ready, key (EXEC) to obtain the Utilities menu.

6.3 EXECUTION ERROR MESSAGES

During the actual execution of the Copy/Verify Utility, error conditions, if encountered, are printed. The next file operation begins without interruption.

MESSAGE	DESCRIPTION
FILE - filename - CANNOT BE COPIED.	This message indicates that the filename shown cannot be copied for one of the following reasons: 1. The file name already exists on the output disk, if ADD output mode. 2. The file name does not exist on the output disk, if REPLACE output mode. 3. There is insufficient disk space in the output file or on the output disk to copy the input file with requested extra sectors.

FILE - filename - DOES NOT VERIFY	This message indicates that the filename shown does not verify. The KFAM-7 file KFAMWORK, a work file, will not verify properly. Otherwise, the operation should be reattempted for any other file name.

CHAPTER 7
THE CREATE REFERENCE FILE UTILITY PROGRAM

7.1 INTRODUCTION

The CREATE REFERENCE FILE creates, modifies, or lists a reference file. Once created, a reference file is used by either the COPY/VERIFY or PROGRAM COMPARE utility program to indirectly specify what disk file will be used in their respective operations. The sequence the file names are entered in during CREATE REFERENCE FILE operation determines their (later) sequence of use by the COPY/VERIFY or PROGRAM COMPARE utility programs. For each copy/verify operation, or each program compare operation, a pair of file names are required. Up to 999 pairs of file names may be processed.

For COPY/VERIFY operations, the name of the file to be copied from the input disk and the file name given to the copied file on the output disk are required for each file to be copied/verified. These file names are hereafter referred to as "input file name" and "output file name." One input file name may be entered more than once, thus the same input file may be copied as several output files. A uniform number of extra (free) sectors may be reserved for each file copied indirectly, if specified during COPY/VERIFY. The reference file must reside on the input disk, which is required during CREATE REFERENCE FILE operation to ensure valid input file name entry.

For PROGRAM COMPARE operations, the name of the program file resident on the first input disk and the name of the program file on the second input disk determine the two program files that are compared. These file names are hereafter referred to as "first file name" and "second file name." One "first file name" may be entered more than once, thus the same program may be compared to more than one program. However, to compare a program to more than one program, it must, of course, reside on the first input disk. Also, the reference file must reside on the first input disk. The first input disk is required during CREATE REFERENCE FILE operation to insure valid first file name entry.

The "create" function creates a new reference file and catalogs it, or reuses a previously-cataloged scratched program or data file. The operator enters the number of file names to be placed in the reference file, and the utility automatically calculates the required file size.

The "modify" function allows modification of an existing reference file's contents by changing, adding, or deleting file name references. It is recommended that the operator have a printed listing of the reference file to be modified.

The "list" function prints a list of all pairs of file names in the specified reference file. Output is to the address specified as the ISS printer address.

A reference file can accommodate 999 file references.

7.2 OPERATING INSTRUCTIONS: CREATE REFERENCE FILE

	DISPLAY	INSTRUCTIONS
1.		1. From the ISS Utilities menu, load CREATE REFERENCE FILE by touching the indicated Special Function Key.
2.	ENTER THE DESIRED OPTION ?-/ OPTIONS AVAILABLE 0 - CREATE 1 - MODIFY 2 - LIST (PRINTER REQUIRED)	2. Enter 0 (zero) to create a new reference file. Enter 1 to modify an existing reference file. Enter 2 to list the contents of an existing reference file.
3.	ENTER THE INPUT ADDRESS. ?---/ ADDRESSES AVAILABLE = XYY	3. Enter the device address of the input disk, which contains the files to be copied for COPY/VERIFY, or is the first input disk for PROGRAM COMPARE.
4.	ENTER THE NAME OF THE REFERENCE FILE ?--------	4. If the selected OPTION is MODIFY or LIST, enter the file name of the reference file to be modified or listed. For the CREATE option, enter a unique file name for the new reference file to be created. Go to step 10, unless the OPTION is CREATE, in which case go to the next step.

NOTE:

If the selected OPTION is CREATE and the entered file name is the name of an existing scratched file, that file's space will be occupied by the new (active) reference file.

5. ENTER THE NUMBER OF FILE NAMES TO BE ENTERED.
?-/

5. Enter the number of pairs of file names to be in the reference file, from 1 to 999. This value can be approximate but must not be less than the actual number needed. You may enter a value that allows for future expansion. This entry determines FILE SIZE of the reference file, which is always in multiples of 14.

NOTE:

The reference file saves 14 file references per sector. Since actual file length varies in one sector increments, the actual file size may be as much as 13 file references larger than that requested.

6. MOUNT INPUT DISK
KEY RETURN(EXEC) TO RESUME.

6. Mount the input disk at the device address entered in step 3. Go to step 7.

An error message in form of "ERROR RETURN CODE FROM OPEN" and either "=S" or "=D" indicates an error opening the reference file. See Table 5-2.

7. WILL ALL OUTPUT NAMES BE THE SAME AS THE INPUT NAMES. (Y/N)
?_

7. If you do not wish to change the names of any of the copied files, enter Y. Otherwise, enter N.

8.	ENTER INPUT FILE NAME. (0 = END) ?--------	8. Enter the name of the file to be copied. If "Y" was answered at step 7, repeat this step; otherwise go to step 9. When the names of all the files to be copied have been included in the reference file, enter zero to end entry and go to step 15. If the input file name entered is not cataloged, or active, the file name entered and NOT AN ACTIVE FILE appear. Reenter a valid file name. If the input file name entered is identical to one previously entered, DUPLICATE INPUT FILE NAME, and KEY RETURN(EXEC) to RESUME appear. To accept the duplicate input file name, key X (EXEC). To reenter the input file name key (EXEC).
9.	INPUT FILE NAME = FILENAME ENTER OUTPUT FILE NAME (EXEC = SAME AS INPUT) ?---------	9. Enter the name to be given to the copied file, or key (EXEC) to use the input name as the output name. Go to step 8. If the output file name entered is identical to one previously entered, the file name entered and IS A DUPLICATE FILE appear. Key (EXEC) and enter a different output file name.
10.	MOUNT INPUT DISK KEY RETURN(EXEC) TO RESUME?	10. Mount the disk containing the reference file to be modified. If OPTION = LIST, go to step 16; otherwise go on to the next step. If the message "FILE NOT ON DISK" appears, an active file with the entered name does not exist on the mounted disk. Either mount the correct disk, or key Special Function Key 31 to return to the ISS utilities menu.

If "ERROR RETURN FROM OPEN='D' KEY RETURN (EXEC) TO RESUME?" appears, this indicates a file disposition conflict, such as file not found, was encountered. Either (1) mount the correct disk and the key (EXEC) to retry, or (2) to return to the ISS menu, enter X, (EXEC).

11. ENTER THE FILE NUMBER TO BE MODIFIED (0 = END)
?_----/

11. Enter the number of the reference file entry that is to be modified. Go to the next step.

If the reference file is now correct, enter zero and go to step 15.

12. ENTER THE NUMBER OF THE DESIRED OPTION
?_/

OPTIONS AVAILABLE

0 - NO CHANGE
1 - MODIFY
2 - DELETE

12. The reference file entry is displayed. For the displayed entry choose from the available options.

Enter 0 to accept the displayed reference file entry. Go to step 11.

Enter 1 to modify the displayed reference file entry. Go to step 13.

Enter 2 to delete the displayed reference file entry. Go to step 11.

13. ENTER THE INPUT FILE NAME
?_--------

13. Enter the name of the file to be copied.

If the input file name entered is not cataloged, or active, the file name entered and NOT AN ACTIVE FILE appear. Reenter a valid file name.

If the input file name entered is identical to one previously entered, DUPLICATE INPUT FILE NAME and KEY RETURN(EXEC) TO RESUME appear. To accept the duplicate input file name, key X (EXEC). To reenter the input file name, key (EXEC).

14.	INPUT FILE NAME = FILE NAME ENTER OUTPUT FILE NAME. (EXEC = SAME AS INPUT) ?--------	14. Enter the name to be given to the copied file, or key (EXEC) to use the input name as the output name. Go to Step 11. If the output file name entered is identical to one previously entered, the file name entered and IS A DUPLICATE FILE appear. Key (EXEC) and enter a different output file name.
15.	DO YOU WANT TO PRINT THE REFERENCE FILE? (Y/N) ?	15. To obtain a listing of the reference file, enter Y and go to the next step. The listing is output to the ISS printer address. Otherwise, enter N and go to step 17.
16.	PRINTING REFERENCE FILE	16. Temporary display. If the file is not being printed, then the printer is not ready. Ready the printer. 8 1/2" by 11" paper is required.
17.	REMOUNT ISS PLATTER IF REMOVED	17. If the ISS disk was removed, remount it at the ISS loading address.

7.3 EXECUTION ERROR MESSAGES

Create Reference File error messages are discussed under the program which later uses the Created Reference File. Other error messages are discussed above in the Operating Instructions. (See Copy/Verify or Program Compare.)

CHAPTER 8
SORT DISK CATALOG UTILITY PROGRAM

8.1 INTRODUCTION

The SORT DISK CATALOG utility program prints a sorted list of the contents of a disk catalog index. The list may be sorted by file entry sequence in the index (index sector sequence), alphabetically by file name, or numerically by starting sector address; it may be output to the display or to the printer. Active files, scratched files, or both, may be included in the list. The size of the sort array limits a single list to 340 files. To process more than 340 files, refer to Section 8.3. During processing, if the array is filled before exhausting the selected index items, a partial list is produced.

The printed output report resembles the LIST DC statement. Operationally, the option to list (1) only active files, (2) only scratched files, and (3) both active and scratched files is provided, and the printed report reflects this choice. The printout includes, for the chosen disk, the number of data and program files in each of the file categories: active, scratched, and total. In addition, the number of sectors USED by files, the number of FREE sectors within files, and the number of sectors allocated to files, for the chosen file category (active, scratched, or both) are printed.

8.2 OPERATING INSTRUCTIONS: SORT DISK CATALOG

DISPLAY	INSTRUCTIONS
1.	1. From the ISS Utilities menu, load the SORT DISK CATALOG utility by touching the specified Special Function Key. NOTE: If the default INPUT ADDRESS is not a valid ISS disk address, the prompt shown in step 5 first appears instead of the prompt shown in step 2, and REENTER appears below the prompt. After completing step 5, go to step 2.
2. ENTER THE NUMBER OF THE DESIRED FUNCTION ?-/ (Functions 1-5 appear with default values for 1-4)	2. To change one parameter, enter its respective number, and go to the step listed below.

TO CHANGE	ENTER	GO TO STEP
SORT OPTION	1	3
FILE TYPE	2	4
INPUT ADDRESS	3	5
OUTPUT DEVICE	4	6

Enter 5 when all parameters are correct. Go to step 7.

3. ENTER THE SORTING OPTION.
?-/

OPTIONS AVAILABLE

1 - SORT BY NAME
2 - SORT BY STARTING SECTOR
3 - SORT BY INDEX SEQUENCE

3. To sort the catalog index into ascending order of the file names, enter 1. To sort the (catalog) index into ascending order of each file's starting sector addresses, enter 2. To sort the index into ascending order based on index sector sequence (same as LIST DC statement), enter 3. Go to step 2 unless CHANGE ALL parameters was chosen.

4. ENTER TYPE OF FILE TO LIST.
?-/

FILE TYPE

0 - ALL
1 - ACTIVE
2 - SCRATCHED

4. Enter 0 (zero) to include all cataloged files in the output list. Enter 1 to include only active files in the output list. Enter 2 to include only scratched files in the output report.
Go to step 2 unless CHANGE ALL parameters was chosen.

5. ENTER THE INPUT ADDRESS
?---/

AVAILABLE ADDRESSES = XYY

5. Enter the device address at which the input disk will be mounted. Valid ISS disk addresses are displayed.
Go to step 2 unless CHANGE ALL parameters was chosen.

6. ENTER THE OUTPUT DEVICE.
(0-CRT, 1-PRINTER)
?-/

6. Enter 0 (zero) to view the sorted index on the CRT screen. Enter 1 to list the sorted index on a printer.

7. DO YOU WISH TO SAVE THESE VALUES AS DEFAULTS. (Y/N)
?-

7. To save the currently displayed parameters as the default values for the SORT DISK CATALOG utility, enter Y. Otherwise, enter N.

8. ENTER TITLE FOR LIST
?----------

8. Enter a title for the sorted index. The title entered will appear on each output page if a printer is used.

9.	MOUNT DISK TO BE LISTED - UNIT XYY KEY RETURN(EXEC) TO RESUME?	9. Mount the input disk to be listed at the displayed XYY address. When ready, key (EXEC).
10.	SCANNING DISK INDEX	10. Temporary display appears while index is being scanned and sorted.
11.		11. The sorted catalog index is displayed or printed. If displayed on the CRT screen, about eight files appear at a time; key (EXEC) to view the next eight files. After all files have been displayed, key (EXEC) to continue. A summary of disk use appears. Key (EXEC) again.
12.	KEY THE NUMBER OF THE DESIRED OPTION. ?_/ OPTIONS AVAILABLE 1 - REPRINT LIST ON CRT 2 - REPRINT LIST ON PRINTER 3 - RETURN TO MENU	12. To view the same sorted index on the screen, key 1 (EXEC). To list the same sorted index on a printer, key 2 (EXEC). To bring the Utilities Menu to the screen, key 3 (EXEC).
13.	REMOUNT ISS PLATTER KEY RETURN(EXEC) TO RESUME?	13. If removed, this prompt appears requesting that the ISS disk be mounted at the ISS loading address. When ready, key (EXEC). The Utilities menu will appear.

8.3 SORT DISK CATALOG MODIFICATIONS

A programmer may increase the maximum number of files (from 340) that can be sorted per operation by the SORT DISK CATALOG utility.

To change the maximum number of files which can be sorted, the following procedures are required.

1. Enter: SELECT DISK xyy (xyy indicates the ISS loading address.)
   CLEAR
   LOAD DC T "ISSA030B"

2. On line 70, change the two array element subscripts in the dimension statements in the following arrays to the subscripts C0 and C2 below: N$(), L$(), L1$().

3. On line 130, set the values of scalar variables C0 and C2 to the subscripts listed below.

   For example, with a 16K partition, edit line 70 so that the array element subscripts for N$(), L$(), and L1$() are N$(2,218)12, L$(2,218)2, L1$(2,218)2, and edit line 130 to C0=2:C2=218.

4. Enter: SCRATCH T "ISSA030B"
   SAVE DC T () "ISSA030B"

Memory	C0	C2	Number of Files
16K	2	218	436
32K	6	224	1464
48K	10	248	2480
64K	14	250	3500

CHAPTER 9
DISK DUMP UTILITY PROGRAM

9.1 INTRODUCTION

The DISK DUMP utility program prints the contents of an entire disk file, or part of a file as defined by physical record number (relative sector) limits. Three kinds of listings can be obtained: Vertical Dump, Horizontal Dump, and Data File Structure Dump.

The Vertical and Horizontal Dumps print the hexadecimal and alphanumeric character equivalents of the file's contents. They differ only in output format. In the Horizontal Dump, the alphanumeric values are given on the same line as the hexadecimal values. In the Vertical Dump, the alphanumeric characters are on one line, with the hexadecimal values given on the two lines immediately below them. Hexadecimal (hex) values below 20 cause "." to be printed in place of an alphanumeric character; hex values above FE print "@".

The third kind of dump is the Data File Structure Dump, which is applicable only to data files not written in BA mode. It prints the contents of a data file, field by field, giving the type of field (numeric or alphanumeric), the length, and the value represented relative to the type of field.

The Vertical Dump requires 8 1/2 by 11 inch printer paper; the other dumps require 11 by 14 inch paper. The ISS date, page number, and file name appear at the top of each printed page. Sector locations and byte numbers are indicated for each record.

9.2 OPERATING INSTRUCTIONS: DISK DUMP

DISPLAY	INSTRUCTIONS
1.	1. From the ISS Utilities menu, load the DISK DUMP utility by touching the specified Special Function Key.

2. ENTER THE DISK ADDRESS. ?-- ADDRESSES AVAILABLE = XYY	2. Enter the device address where the input disk will be mounted. If REENTER appears before any entry is made, the default value is not an ISS disk address and must be corrected. Valid ISS disk addresses are displayed.
3. MOUNT INPUT PLATTER KEY RETURN(EXEC) TO RESUME?	3. Mount the disk containing the file to be dumped at the disk address entered in step 2. When ready, key (EXEC).
4. ENTER THE NAME OF THE FILE TO BE DUMPED. ?--------	4. Enter the name of the disk file whose contents will be dumped. If the file name entered could not be located in that disk's catalog index, ERROR - FILENAME - FILE IS NOT AN ACTIVE FILE appears. Reenter the file name correctly.
	NOTE: If a multistation file with Password protection is being dumped, a prompt requesting Password entry will appear after step 4. If so, enter this file's Password exactly as previously assigned. "ERROR RETURN CODE = P" appears if it is incorrectly entered. Reenter the Password correctly (or key S.F. Key 31).

5.	PHYSICAL RECORDS WITHIN FILE TO BE DUMPED. (FFFFF-LLLLL OR ALL) ?--------	5. To dump the entire file, enter ALL. To dump a portion of the file, enter the beginning and ending physical record (relative sector) numbers, relative to 00000; include a hyphen between the two entries. To dump physical records 102 to 105 for example, enter 101-104. Record number limits are included in the dump. Physical record numbers are based on DC or DA Load/Save access commands. BA access files must be dumped using DISK DUMP 3 ALL option; that is, physical record numbers entered for BA access files will cause unpredictable results.
6.	ENTER TYPE OF DUMP, SEE TABLE BELOW. ?-/ -- DUMPS AVAILABLE -- 1 - HORIZONTAL 2 - VERTICAL 3 - DATA FILE STRUCTURE	6. Enter 1 to choose a Horizontal dump. Enter 2 to choose a Vertical dump. For data files only, enter 3 to choose a Data File Structure dump. (Displayed for data files only.) Only DC or DA Load/Save files are valid; BA files must not be used by a Data File Structure dump.
7.		7. The report is printed. Key H to interrupt printing.
8.	ENTER THE NUMBER OF THE DESIRED OPTION. ?-/ -- INTERRUPT OPTIONS -- 1. CONTINUE 2. MODIFY 3. RESTART PROGRAM 4. RETURN TO MENU	8. The step 8 prompt appears if (1) printing terminates normally, or (2) if a printer problem or H key caused an interrupt. Enter 1 to continue interrupted printing; the step 8 prompt will reappear. Enter 2 to dump a portion or all of the same file; go to step 5. Enter 3 to restart DISK DUMP and dump a file on the same or different disk; go to step 2. Enter 4 to obtain the Utilities menu. Remount the ISS platter and key (EXEC) if REMOUNT ISS PLATTER KEY RETURN(EXEC) TO RESUME is displayed.

CHAPTER 10
DECOMPRESS UTILITY PROGRAM

10.1 INTRODUCTION

The DECOMPRESS utility program breaks up all multistatement lines in a program so that each statement appears on a numbered line by itself. As input it accepts any cataloged program file, selected program files, all files on the input disk, or all files within specified alphabetical limits. It outputs the decompressed version on the output disk as a cataloged program file (or files).

Decompress breaks up multistatement lines by assigning to each BASIC statement, after the first in a line, a line number one greater than that of the previous statement in the line.

If there are not enough line numbers available between two lines in the input program, the utility decompresses until it runs out of line numbers. A multistatement line is left at the highest numbered line in such a group. When encountered, this condition is brought to the operator's attention.

In producing the output file, the utility creates a uniform system of indentation:

a) All REM statements are indented one space from the line number.

b) All other statements, except those within a FOR...NEXT loop, are indented 5 spaces.

c) Non-REM statements that are within a FOR...NEXT loop are indented 2 spaces in addition to any indentation they would have otherwise received.

If selected files are processed, the maximum number of files is 40. Any number of files may be processed under the ALL or RANGE option.

10.2 OPERATING INSTRUCTIONS: DECOMPRESS

DISPLAY	INSTRUCTIONS
1.	1. From the ISS Utilities menu, load DECOMPRESS by touching the indicated Special Function Key.
	NOTE:
	If the default INPUT ADDRESS or default OUTPUT ADDRESS is not a valid ISS disk address, the prompts shown respectively in step 5 or step 6 will appear instead of the prompt in step 2; also, REENTER appears below the prompt. After completing step 5, step 6, or both, go to step 2.
2. ENTER THE NUMBER OF THE DESIRED FUNCTION ?-/ (Functions 1-6 appear with default values for 1-4)	2. To change a parameter, enter its respective number and go to the step listed below. To exchange the input and output addresses, enter 5.

TO CHANGE	ENTER	GO TO STEP
MODE	1	3
EXTRA SECTORS	2	4
INPUT ADDRESS	3	5
OUTPUT ADDRESS	4	6

Enter 6 when all parameters are correct. Go to step 7.

3. ENTER THE DESIRED MODE (1-ALL, 2-PART or 3-RANGE) ?-/	3. Enter 1 to decompress all files on the input disk. Enter 2 to decompress files on the input disk whose file names will be entered later. Enter 3 to decompress files on the input disk whose file names fall within alphabetic limits specified later. Go to step 2.

4.	ENTER THE NUMBER OF EXTRA SECTORS (MUST BE =2) ?--/	4. Enter the number of sectors to be included in each file, in addition to those required to save the file. Must be greater than or equal to 02, up to 99.
5.	ENTER THE INPUT ADDRESS ?--- ADDRESSES AVAILABLE = XYY	5. Enter the device address at which the input disk will be mounted. Valid ISS disk addresses are displayed. Go to step 2.
6.	ENTER THE OUTPUT ADDRESS ?--- ADDRESSES AVAILABLE = XYY	6. Enter the device address at which the output disk will be mounted. Valid ISS addresses are displayed. The output disk address must not be the same as the input disk address.
7.	DO YOU WISH TO SAVE THESE VALUES AS DEFAULTS. (Y/N) ?-	7. To save the currently displayed parameters as the default values for the DECOMPRESS Utility, enter Y. Otherwise, enter N. If MODE is: ALL go to step 12 PART go to step 8 RANGE go to step 10
8.	MOUNT INPUT PLATTER KEY RETURN(EXEC) TO RESUME?	8. Mount the input disk at the displayed input address. (The ISS disk may be removed if necessary.) When ready, key (EXEC).

9.	ENTER THE NAME OF FILE #N. (0 = END) ?--------	9. Enter the name of the Nth input file to be decompressed. Repeat step 9 until all files have been entered, then enter 0 (zero) to end entry and go to step 12. The file name entered and NOT AN ACTIVE PROGRAM FILE appears if the file name entered is not cataloged, not active, or not a program file. Reenter a valid program file name. The file name entered and IS A DUPLICATE FILE NAME appears if the file name entered is identical to a file name previously entered. Reenter a different file name.
10.	ENTER LOWER LIMIT OF FILE NAMES ?--------	10. Enter the lower alphabetic limit of the file names to be decompressed. The lower limit and upper limit entries need not be file names, but the lower limit must be alphabetically less than the upper limit, else INVALID RANGE and REENTER appears (in step 11). File names entered as limits are included in decompression.
11.	ENTER THE UPPER LIMIT OF FILE NAMES ?--------	11. Enter the upper alphabetic limit of the file names to be decompressed.
12.	REMOUNT ISS PLATTER KEY RETURN(EXEC) TO RESUME?	12. If removed, this prompt appears requesting that the ISS disk be mounted at the ISS loading address. When ready, key (EXEC).
13.	MOUNT PLATTERS AT THE INDICATED ADDRESSES. KEY RETURN(EXEC) TO RESUME?	13. Mount the input and output disks at the displayed addresses. (The ISS disk may be removed if necessary.) When both disks are ready, key (EXEC).

14.	DECOMPRESSING FILE NUMBER = NNN FILE NAME = FILENAME	14. This processing display shows the file currently being decompressed. After all files have been decompressed, the Utilities menu will appear unless the ISS disk was removed in step 13. If removed, "MOUNT ISS PLATTER, KEY RETURN(EXEC) TO RESUME?" appears. Remount the ISS disk at the ISS loading address. When ready, key (EXEC). The Utilities menu will appear.

10.3 EXECUTION ERROR MESSAGES

During execution of the Decompress Utility, error conditions are printed. The next file operation begins without interruption.

MESSAGE	DESCRIPTION
FILE - FILENAME - CANNOT BE DECOMPRESSED	This file, whose filename is shown, cannot be Decompressed because either (1) insufficient disk space exists for the decompressed output file, or (2) the file cannot be accessed because it is protected.
DECOMPRESSION WAS INCOMPLETE FOR FILE - FILENAME.	This message results if there were insufficient line numbers between two lines in a program to assign a unique line number to each statement. (If desired, the program can be renumbered, saved, and decompressed again to complete the decompression.)

CHAPTER 11
LIST/CROSS-REFERENCE UTILITY PROGRAM

11.1 INTRODUCTION

The LIST/CROSS-REFERENCE utility program consists of two components which may be executed independently or sequentially under program control.

The LIST component breaks up all the multistatement lines of a program, printing each BASIC-2 statement on a separate line.

For example, the statement line

```
410COM F$1,T$1,N$(64)8,F,F1,C,O:COM W4$1,Q6$64,D$1,D$(2)3:COM L,
E,E1:DIM N$8,B$(16),L$1,H$2,W1$8:GOTO 660
```

is listed as:

```
410 COM F$1,T$1,N$(64)8,F,F1,C,O
    :COM W4$1,Q6$64,D$1,D$(2)3
    :COM L,E,E1
    :DIM N$8,B$(16),L$1,H$2,W1$8
    :GOTO 660
```

The CROSS-REFERENCE component assembles and prints four cross-reference tables: a line number cross-reference, a variable cross-reference, location of marked subroutines, and a marked subroutines cross-reference.

In the line number cross-reference table, each line number referenced in the program is printed, together with the numbers of the lines that contain the references. The variable cross-reference lists each variable that appears in the program and identifies the lines in which it appears.

Note that BASIC-2 Language statements and global variables applicable only to the 2200MVP Central Processor, e.g., $OPEN, $RELEASE TERMINAL, SELECT @PART, are not recognized as statements by LIST/CROSS-REFERENCE.

NOTE:

In the variable cross-reference there are certain BASIC-2 statement forms which cause a nonvariable to be referenced as a variable; there are others which cause array variables to be referenced as scalar variables. The BASIC-2 statements in which a nonvariable can appear that is referenced as a variable are ADD, ADD C, AND, XOR, BOOL, INIT, $TRAN, $GIO, PLOT, DATASAVE BT, and DATALOAD BT. Array variables in the matrix statements are referenced as scalar variables. For a complete specification of the conditions for these occurrences, see Appendix C.

The location of all marked routines is the location of the DEFFN' statements. The marked subroutine cross-reference table lists all the GOSUB' references to DEFFN' marked subroutines.

The date, file name, and page number appear atop each page of output.

During the program inspection stage of the CROSS-REFERENCE utility, an internal table is built up as variables, subroutines, and line references are encountered. Should this internal table be filled before the entire program has been inspected, the utility prints the four cross-reference tables, clears the internal table, and resumes program inspection from the point at which it left off. The final result is two sets of partial cross-reference tables with each set complete for the program section inspected.

Input programs for the LIST/CROSS-REFERENCE utility are assumed to be free of syntax errors.

All the program files on the input disk or all program files within alphabetic limits may be processed. Up to 40 selected files may be processed.

Available with the LIST option (not LIST/CROSS-REFERENCE), use of the statement REM % must obey the following conventions:

1. To supply two blank lines following the REM % statement and then print an expanded size title, the following format is necessary:

```
REM % TITLE
     ↑
     |
ONE BLANK
BETWEEN % AND
FIRST CHARACTER
OF THE TITLE.
```

2. To advance the carriage return to the top of the next page (forms feed), and then print the expanded size title at the top of that page, the following format is necessary:

```
REM %↑TITLE
     ↑
NO BLANKS
BETWEEN ↑
(UPWARD ARROW)
AND FIRST
CHARACTER IN
THE TITLE.
```

In both cases, the first character of the title must be provided and be in the positions indicated above.

11.2 OPERATING INSTRUCTIONS: LIST/CROSS-REFERENCE

DISPLAY	INSTRUCTIONS
1.	1. From the ISS Utilities menu, load LIST/CROSS-REFERENCE by touching the indicated Special Function Key.
2. ENTER THE NUMBER OF THE DESIRED FUNCTION. ?-/ (FUNCTIONS 1-7 ARE DISPLAYED WITH DEFAULT VALUES FOR 1-6)	2. To change a parameter, enter its selection number and go to the step listed below. TO CHANGE — GO TO STEP FUNCTION — 3 INPUT ADDRESS — 4 MODE — 5 MARGIN — 6 LINE LENGTH — 7 LINES/PAGE — 8 When all displayed parameters are correct, enter 7 and go to step 9.
3. ENTER THE NUMBER OF THE OPERATION DESIRED. ?-/ FUNCTIONS AVAILABLE 1 - LIST 2 - CROSS-REFERENCE 3 - LIST/CROSS-REFERENCE	3. If only a listing is required, enter 1. If only a cross-reference is required, enter 2. If both a listing and a cross-reference is required, enter 3. Go to step 2.

4. ENTER THE INPUT ADDRESS ?--- ADDRESSES AVAILABLE = XYY	4. Enter the device address at which the disk will be mounted. Valid ISS disk addresses are displayed. Go to step 2.
5. ENTER THE DESIRED MODE (1-ALL, 2-PART OR 3-RANGE) ?---/	5. To list/cross-reference all files on the disk, enter 1. To list/cross-reference some files on the disk whose names will be entered, enter 2. To list/cross-reference those files whose names fall between alphabetic limits specified later, enter 3. Go to step 2.
6. ENTER THE NUMBER OF SPACES FOR THE MARGIN. (1-10) ?--/	6. Enter the number of spaces indented from the left margin from 01 to 10. Go to step 2.
7. ENTER THE LINE LENGTH, INCLUDING MARGIN (70-128) ?---/	7. Including the number of spaces entered in reply to step 6, enter the total line length from 070 to 128. Go to step 2.
8. ENTER THE NUMBER OF LINES PER PAGE. (10-55) ?--/	8. Enter the total number of lines per printed page from 10 to 55. This value determines when a top of form occurs during printing. Go to step 2.
9. DO YOU WISH TO SAVE THESE VALUES AS DEFAULTS. (Y/N) ?-	9. To save the displayed values as default parameters for LIST/CROSS-REFERENCE, enter Y. Otherwise, enter N. If the MODE is: ALL go to step 14. PART go to step 10. RANGE go to step 12.
10. MOUNT INPUT PLATTER KEY RETURN(EXEC) TO RESUME?	10. Mount the input disk at the displayed INPUT ADDRESS. When ready, key (EXEC).

11.	ENTER THE NAME OF FILE #N (0 = END) ?--------	11. Enter the name of the Nth file to be listed/cross-referenced. The file name must be that of an active cataloged program file, or else FILE - FILENAME - NOT AN ACTIVE PROGRAM FILE appears. Reenter the file name. Repeat step 11 until all file names have been entered, then enter 0 (zero) to end entry and go to step 14.
12.	ENTER LOWER LIMIT OF FILE NAMES ?--------	12. Enter the lower alphabetic limit of the file names to be listed/cross-referenced. The lower limit and upper limit entries need not be file names, but the lower limit must be alphabetically less than the upper limit, else INVALID RANGE and REENTER appear (in step 13). File names entered as limits are included in list/cross-reference.
13.	ENTER UPPER LIMIT OF FILE NAMES ?--------	13. Enter the upper alphabetic limit of file names to be listed/cross-referenced. This temporary display requires no operator action.
14.	INITIALIZING TABLES FOR NEXT MODULE	14. If removed, a prompt appears requesting that the ISS disk be remounted at the ISS loading address. When ready, key (EXEC).
15.	MOUNT INPUT PLATTER KEY RETURN(EXEC) TO RESUME?	15. Mount the input disk at the displayed input address. When ready, key (EXEC).

16.	PROCESSING FILE N NAME = XXXXXXXX	16. This display remains throughout processing. If H is keyed, processing stops at the end of the current file. If P is keyed, processing stops at the end of the current page of output. With processing stopped, keying RETURN causes processing to resume. If a cross-reference file was selected for output, the additional message "S.F. 1-REPRINT CROSS-REFERENCE FILE" appears. If Special Function key 1 is depressed with processing halted, the cross-reference tables for the current file are printed a second time.

11.3 EXECUTION ERROR MESSAGES

During execution of the List/Cross-Reference Utility, error conditions are printed. The next file operation begins without interruption.

MESSAGE	DESCRIPTION
FILE - filename - PROTECTED	Protected file encountered and skipped.

CHAPTER 12
COMPRESS UTILITY PROGRAM

12.1 INTRODUCTION

The COMPRESS utility program reduces the amount of memory occupied by an application program. In addition to permitting the execution of more powerful programs, COMPRESS increases the speed of program execution, and reduces storage requirements.

The COMPRESS utility does three things to an input program file:

a) It eliminates all REM statements, except those in the first statement line.

b) It eliminates all space characters not enclosed by quotation marks.

c) It eliminates as many unnecessary line numbers as possible by assigning to each line number the maximum number of BASIC-2 statements consistent with program operation.

> CAUTION:
>
> A program to be compressed cannot contain branches to statement lines beginning with a REM statement, since all such lines (except the first line in the program) are deleted. Also, program text should be compressed only once, unless it has been decompressed since it was last compressed.

The compress utility works in two stages. In the first stage the input program is examined, and a table is built of all line numbers referred to by statements in the program. This table is called the "protect table", since, if the program is to execute properly, these referenced line numbers must be preserved.

A convention is observed that allows the programmer to explicitly exempt any statement line from being compressed into another line. A blank REM statement appearing within an input program causes the next non-REM line to be protected. That is, blank REM statements are not attached to preceeding or following lines. Blank REM statements which immediately surround a single line therefore have the effect of exempting that line from compression. (A blank REM statement is REM followed immediately by RETURN(EXEC).) If a compressed program is compressed a second time, lines previously protected by blank REM's are no longer protected, since the protecting REM's have been eliminated.

The first statement line in a program is unaltered, regardless of its content.

During the utility's second stage, called "compression", the compressed version of the program is produced and written to the output disk.

The output disk:

a) Must have a catalog established on it.

b) Must not have a file with the same name as the input program file.

c) Must have sufficient space to store the input program file in its precompressed state.

The utility compresses selected files, all files within alphabetical limits, or all files from the input disk. However, the maximum number of selected files for a single execution is 40. The input disk files must be cataloged and active.

12.2 OPERATING INSTRUCTIONS: COMPRESS

DISPLAY	INSTRUCTIONS
1.	1. From the ISS Utilities menu, load COMPRESS by touching the indicated Special Function Key.
	NOTE:
	If the default INPUT ADDRESS or default OUTPUT ADDRESS is not a valid ISS disk address, the prompts shown respectively in step 5 or step 6 will appear instead of the prompt in step 2; also, REENTER appears below the prompt. After completing step 5, step 6, or both, go to step 2.
2. ENTER THE NUMBER OF THE DESIRED FUNCTION. ?_/ (Functions 1-6 appear with default values for 1-4)	2. To change a parameter, enter its respective number and go to the step listed below. To exchange the input and output addresses, enter 5.

TO CHANGE	ENTER	GO TO STEP
MODE	1	3
OPTION	2	4
INPUT ADDRESS	3	5
OUTPUT ADDRESS	4	6

Enter 6 when all parameters are correct. Go to step 7.

3. ENTER THE DESIRED MODE (1-ALL, 2-PART OR 3-RANGE) ?---/	3. Enter 1 to compress all files on the input disk. Enter 2 to compress files on the input disk whose file names will be entered later. Enter 3 to compress files on the input disk whose file names fall within alphabetic limits specified later. Go to step 2.

4. ENTER THE DESIRED COMPRESSION OPTION. (1-180 OR 2-256) ?-/	4. Enter 1 to implement a compression option of 180 bytes per line. Enter 2 to implement a compression option of 256 bytes per line. Go to step 2.
5. ENTER THE INPUT ADDRESS ?---/ ADDRESSES AVAILABLE = XYY	5. Enter the device address at which the input disk will be mounted. Valid ISS disk addresses are displayed. Go to step 2.
6. ENTER THE OUTPUT ADDRESS ?---/ ADDRESSES AVAILABLE = XYY	6. Enter the device address at which the output disk will be mounted. Go to step 2.
7. DO YOU WISH TO SAVE THESE VALUES AS DEFAULTS. (Y/N) ?-/	7. To save the currently displayed parameters as the default values for the COMPRESS Utility, enter Y. Otherwise, enter N. If MODE is: ALL go to step 13 PART go to step 8 RANGE go to step 10
8. MOUNT INPUT PLATTER KEY RETURN(EXEC) TO RESUME?	8. Mount the input disk at the displayed input address. (The ISS disk may be removed if necessary.) When ready, key (EXEC).
9. ENTER THE NAME OF FILE #N. (0 = END) ?--------	9. Enter the name of the Nth input file to be compressed. Repeat step 9 until all file names have been entered, then enter 0 (zero) to end entry and go to step 12. The file name entered and NOT AN ACTIVE PROGRAM FILE appears if the file name entered is not cataloged, not active, or not a program file. Reenter a valid program file name.

		The file name entered and IS A DUPLICATE FILE NAME appear if the file name entered is identical to a file name previously entered. Reenter a different file name.
10.	ENTER LOWER LIMIT OF FILE NAMES. ?--------/	10. Enter the lower alphabetic limit of the file names to be compressed. The lower limit and upper limit entries need not be file names, but the lower limit must be alphabetically less than the upper limit, else INVALID RANGE and REENTER appear (in step 11). File names entered as limits are included in compression.
11.	ENTER THE UPPER LIMIT OF FILE NAMES ?--------/	11. Enter the upper alphabetic limit of the file names to be compressed.
12.	REMOUNT ISS PLATTER KEY RETURN(EXEC) TO RESUME?	12. If removed, this prompt appears requesting that the ISS disk be mounted at the ISS loading address. When ready, key(EXEC).
13.	MOUNT PLATTERS AT THE INDICATED ADDRESSES. KEY RETURN(EXEC) TO RESUME?	13. Mount the input and output disks at the displayed addresses. (The ISS disk may be removed if necessary.) When both disks are ready, key (EXEC).
14.	BUILD PROTECT TABLE FOR FILE N TOTAL BLOCKS = NNNN FILE NAME = XXXXXXXX BLOCKS READ = NNNN	14. Temporary display appears during processing.

15.	COMPRESSING FILE NUMBER = NNN FILE NAME = FILENAME	15. This processing display shows the file currently being compressed. After all files have been compressed, the Utilities menu will appear unless the ISS disk was removed in step 13. If removed, "MOUNT ISS PLATTER, KEY RETURN(EXEC) TO RESUME?" appears. Remount the ISS disk at the ISS loading address. When ready, key (EXEC). The Utilities menu will appear.

12.3 EXECUTION ERROR MESSAGES

During execution of the Compression Utility, error conditions are printed. The next file operation begins without interruption.

MESSAGE	DESCRIPTION
FILE - FILENAME - CANNOT BE COMPRESSED	One of the following conditions cause this message to occur: a. Output file already exists. b. Insufficient space on disk for file. c. Input file is protected (e.g., SAVE DC FP "___" or SAVE DC F! "___").

CHAPTER 13
RECONSTRUCT INDEX UTILITY PROGRAM

13.1 INTRODUCTION

The RECONSTRUCT INDEX utility program is an aid to the recovery of disk files in the event of accidental destruction of the disk catalog index. The utility searches the disk, looking for file control sectors established during catalog operations. Based on the control sectors found, it attempts to reconstruct a catalog index for the files on the disk, which takes considerable time, especially if a disk has not been reformatted and many files have been scratched.

> CAUTION:
>
> Before executing this utility a backup copy of the disk must be made, because this utility writes on the disk being used. The use of this program is a last resort in recovery procedures. Its success is dependent entirely on the nature of the disk, and thus is not guaranteed to reconstruct the disk index.

The utility constructs names for all data files and for duplicate program file names. The constructed names have the following form:

/*XXXX*/

where: XXXX is a four-digit number. Numbers are assigned consecutively to files that require constructed names, the same as Copy Tape To Disk.

13.2 OPERATING INSTRUCTIONS: RECONSTRUCT INDEX

	DISPLAY		INSTRUCTIONS
1.		1.	From the ISS Utilities menu, load the RECONSTRUCT INDEX utility by touching the indicated Special Function Key.
2.	ENTER THE INPUT ADDRESS ?--- ADDRESSES AVAILABLE = XYY	2.	Enter the device address of the disk whose catalog index is to be reconstructed. Valid ISS disk addresses are displayed.
3.	ENTER THE HIGHEST SECTOR ADDRESS OF THE DISK. ?-----/	3.	Enter the highest sector address at which files are known to exist. If this information is not available, enter the highest sector address of the disk.
4.	ENTER THE NUMBER OF INDEX SECTORS (0=UNKNOWN) ?-----/	4.	Enter the number of sectors in the original index and go to step 8. Enter zero if this is unknown, and go to the next step.
5.	MOUNT INPUT DISK KEY RETURN(EXEC) TO RESUME?	5.	Mount the disk whose index is to be reconstructed. If number of index sectors is unknown, go to the next step. Otherwise go to step 9.
6.	RECONSTRUCTING DISK INDEX	6.	Temporary display; no action required.
7.	REMOUNT ISS PLATTER. KEY RETURN(EXEC) TO RESUME	7.	Remount the ISS disk if it has been removed.
8.	MOUNT INPUT DISK KEY RETURN(EXEC) TO RESUME?	8.	Mount the disk whose index is to be reconstructed.
9.	FILE # FILE NAME START N XXXXXXXX NN END USED NN NN	9.	Processing display (remains on screen while the entire disk's contents are being reconstructed).
10.	REMOUNT ISS PLATTER KEY RETURN(EXEC) TO RESUME	10.	Execution is complete. Remount the ISS disk at the ISS loading address. Key (EXEC) to return to ISS Utilities menu.

CHAPTER 14
FILE STATUS REPORT UTILITY PROGRAM

14.1 INTRODUCTION

The FILE STATUS REPORT utility program provides several functions tailored to multistation disk environments. The program's structure enables some or all functions to be performed without the need for program reload. FILE STATUS REPORT accesses the end catalog trailer record maintained by the Multistation Open/End/Close subroutines used by ISS utility programs and user-written programs (access table). See Chapter 32 for further explanation. Functions performed by FILE STATUS REPORT are summarized below:

1. Close all files open to a particular station. This function closes all open files on a disk which might be left open by operator accidents, during initial program testing, or due to a power failure. Also recommended for end-of-day processing shutdown of that station. In the event of a power failure, the function should be executed for each station.

2. Close a file open to a particular station. Similar to above but allows a specific file to be closed without affecting other files.

3. List the status of all files. This function lists, for each file on a disk, whether the file is currently open or closed. If open, the station numbers currently accessing that file are provided, as well as the access mode for each station. Station access mode status numbers are: 1-Inquiry, 2-Read Only, 3-Shared, and 4-Exclusive.

4. List the status of one file. Similar to above but applies to only one file.

5. List all files currently open to a particular station. Provides a listing of files open to a particular station, which can be useful during initial program test and debugging.

CAUTION:

The File Status Report Utility, functions 1 and 2 above, should be used with extreme caution by programmers knowledgeable of multistation file operation. The option to close one or all files open to a station must be performed when processing for all stations has been terminated, as this will clear all entries in the access table for all stations. For example, if one station were updating records with Exclusive access, and the access table were blanked, another station doing similar updates also with Exclusive access would be granted access to the file, thus two stations would have the file open asumming Exclusive access. The file's updated contents would be erroneous.

NOTE:

With KFAM files, do not run the FILE STATUS REPORT to close files by clearing the access table, except for KFAMWORK, a work file. Instead use the KFAM Utility, RESET ACCESS TABLE, which clears both the access table and KDR of the KFAM File left open accidentally.

14.2 OPERATING PROCEDURES: FILE STATUS REPORT

	DISPLAY	INSTRUCTIONS
1.		1. Load the FILE STATUS REPORT utility by touching the indicated Special Function Key on the ISS Utilities menu.
2.	ENTER THE ADDRESS OF THE DATA DISK ?---/ ADDRESSES AVAILABLE = XYY	2. Enter the device address where the disk to be examined will be mounted. Valid ISS disk addresses are displayed.
3.	MOUNT INPUT DISK UNIT - XYY KEY RETURN(EXEC) TO RESUME?	3. Mount the disk to be examined at the disk address displayed.
4.	ENTER THE NUMBER OF THE DESIRED FUNCTION. ?-/ OPTIONS AVAILABLE	4. Enter 1 to close all files currently open to a station on this disk. Go to step 5. Enter 2 to close one file currently open to a station on this disk.

1 - CLOSE ALL FILES OPEN BY A STATION
2 - CLOSE FILE OPEN BY A STATION
3 - LIST STATUS OF ALL FILES
4 - LIST STATUS OF A FILE
5 - LIST ALL FILES OPEN BY A STATION
6 - CHANGE DISK ADDRESS
(CURRENT ADDRESS = XYY)
7 - RETURN TO MENU

Enter 3 to list the status of all files on this disk.
Enter 4 to list the status of one file on this disk.
Enter 5 to list all files currently open to a station, on this disk.
Enter 6 to change the disk address.
Enter 7 to obtain the Utilities menu. The ISS disk may have to be remounted.
Asterisks appear to the left of the option chosen.

If the LIST option is chosen (key 3, 4, or 5) and the printer is not ON and SELECTed, the message: PRINTER REQUIRED FOR SPECIFIED OPTION, KEY RETURN (EXEC) TO RESUME?" appears. The operator may either (1) make the printer ready or (2) enter X, (EXEC) which returns the prompt in step 4 to the screen.

ENTRY	GO TO STEP
1	5
2	5
3	7
4	6
5	5
6	2

5. ENTER THE STATION NUMBER
?--/

5. Enter station number. If the reply to step 4 was:

1, go to step 7.
2, go to step 6.
5, go to step 7.

6. ENTER THE NAME OF THE FILE
?--------

6. Enter the name of the file to be closed or whose status will be listed. If that file is not an active cataloged file, REENTER appears. Enter a valid file name.

7. SCANNING INDEX FOR FILE NAME

7. Temporary display appears while searching for file(s). Go to step 4 upon completion.

CHAPTER 15
PROGRAM COMPARE UTILITY PROGRAM

15.1 INTRODUCTION

The PROGRAM COMPARE utility program allows a detailed comparison to be made between two programs. The two programs to be compared must be disk files residing on different disks. Statements with the same number from each program file are examined one at a time. PROGRAM COMPARE ignores all remarks and blanks not part of an image statement (%) or literal data string. For example, PROGRAM COMPARE considers the following statements identical:

```
10    REM THIS IS A REMARK:  A=B:  REM ANOTHER REMARK
10A = B
```

As statements of like numbers are compared, the following messages are printed if the conditions described below occur:

MESSAGE PRINTED	CONDITION
#### DO NOT MATCH	Statement number #### occurs in both programs, but their content is not the same.
####DOES NOT EXIST IN FILENAME (XYY)	Statement #### in the file FILENAME at disk address XXY does not exist, whereas #### exists in the other program file.
####FILE FILENAME (XYY) ENDS, BUT FILE NAMEFILE (YXX) CONTINUES	The file FILENAME at disk address XYY ends at statement ####, whereas the file NAMEFILE at disk address YXX continues after statement ####..

####BOTH FILES END	The two files being compared end at statement ####.
PROGRAMS COMPARE - ARE SAME	The two files being compared are identical. They are in fact the same program.

Up to 100 pairs of selected program files may be compared directly, or 999 indirectly, using a reference file created by the Create Reference File Utility. Any number may be compared if ALL or RANGE is specified as the input mode.

15.2 OPERATING INSTRUCTIONS: PROGRAM COMPARE

DISPLAY	INSTRUCTIONS
1.	1. From the ISS Utilities menu, load the PROGRAM COMPARE utility by touching the indicated Special Function Key.

NOTE:

If default values for the INPUT ADDRESS 1 and INPUT ADDRESS 2 are not valid ISS disk addresses, the prompts shown in step 3 or step 5 will appear instead of the step 2 prompt; also, REENTER will appear below the prompt. After completing step 3, step 5, or both, go to step 2.

2. ENTER THE NUMBER OF THE DESIRED FUNCTION.
?-/

(FUNCTIONS 1-5 ARE DISPLAYED WITH DEFAULTS FOR 1-3)

2. To change a parameter, enter its respective number and go to the step listed below. To exchange the input addresses, enter 4.

TO CHANGE	ENTER	GO TO STEP
INPUT ADR 1	1	3
MODE	2	4
INPUT ADR 2	3	5

Enter 5 when all parameters are correct. Go to step 6.

3. ENTER THE FIRST INPUT ADDRESS
?---

ADDRESSES AVAILABLE = XYY

3. Enter the device address at which the first input disk will be mounted. Valid ISS disk addresses are displayed. Go to step 2.

4. ENTER THE NUMBER OF THE DESIRED INPUT MODE.
?-/

INPUT MODES AVAILABLE
1 - ALL 3 - RANGE
2 - PART 4 - INDIRECT

4. To compare all files on the first input disk to any files with the same name on the second input disk, enter 1.
To compare only files whose file names will be entered later, enter 2.
To compare only files whose file names fall within alphabetic limits specified later, enter 3.
To compare files specified indirectly by a reference file, which must reside on the first input disk, enter 4.
Go to step 2.

5. ENTER THE SECOND INPUT ADDRESS
?---

ADDRESSES AVAILABLE = XYY

5. Enter the device address at which the second input disk will be mounted. Valid ISS disk addresses are displayed. Go to step 2.

6. DO YOU WISH TO SAVE THESE VALUES AS DEFAULTS? (Y/N)
?-

6. To save the currently displayed parameters as the default values for PROGRAM COMPARE, enter Y. Otherwise enter N. If input mode is RANGE, go to step 10; otherwise go to the next step.

7.	MOUNT INPUT PLATTERS KEY RETURN(EXEC) TO RESUME?	7. Mount the input disks at the displayed input address. (The ISS disk may be removed if necessary.) If MODE is: ALL, go to step 15, unless the ISS disk was removed, (go to step 13). PART, go to step 8. INDIRECT, go to step 12.
8.	ENTER THE NAME OF THE FIRST INPUT FILE OF PAIR #N (0 = END) ?--------	8. Enter the first file name of the Nth pair of program files to be compared. (First file of a pair resides on first input disk.) Go to step 9. Enter 0 to terminate file name entry and go to step 13 if the ISS disk was removed.
9.	ENTER THE NAME OF THE SECOND INPUT FILE OF PAIR #N (EXEC = SAME AS FIRST). ?--------	9. Enter the second file name of the Nth pair of program files to be compared. (Second file of a pair resides on second input disk.) The corresponding first file name is displayed. Go to step 8. If the file name entered is not an active file, the file name entered, DOES NOT EXIST, and KEY RETURN(EXEC) TO RESUME? appear. Key (EXEC) and reenter the second file name.
10.	ENTER THE LOWER LIMIT OF FILE NAMES ?--------	10. Enter the lower alphabetic limit of the file names on the first input disk to be compared with files of the same name on the second input disk. The lower limit and upper limit entries need not be file names, but the lower limit must be alphabetically less than the upper limit, or else INVALID RANGE and REENTER appear (in step 11). File names entered as limits are included in comparison.

11. ENTER THE UPPER LIMIT OF FILE NAMES ?--------	11. Enter the upper alphabetic limit of the file names to be compared. Go to step 13 if the ISS disk was removed; otherwise go to step 15.
12. ENTER THE NAME OF THE REFERENCE FILE ?--------	12. Enter the file name of the reference file providing the file names to be compared. NOTE: The reference file must presently reside on the first input disk. If the file does not exist, FILE-filename-DOES NOT EXIST ON DEVICE XYY appears. Reenter the reference file name.
13. REMOUNT ISS PLATTER KEY RETURN(EXEC) TO RESUME?	13. If removed, this prompt appears requesting that the ISS disk be remounted at the ISS loading address. When ready, key (EXEC).
14. REMOUNT INPUT PLATTER AT DEVICE XYY.	14. If removed, this prompt appears requesting that the input disk be remounted at the displayed address. When ready, key (EXEC).
15. COMPARING FILE PAIR #N FIRST FILE NAME = FILENAME SECOND FILE NAME = FILENAME	15. The processing display indicates the two program files currently being compared. With the ALL or RANGE mode, SCANNING INDEX FOR FILE NAME first appears. After all pairs of files have been compared, the Utilities menu will appear unless the ISS disk was removed.
16. MOUNT ISS PLATTER , KEY RETURN(EXEC) TO RESUME?	16. Mount the ISS disk at the ISS loading address if removed. When ready, key (EXEC) to obtain the Utilities menu.

15.3 EXECUTION ERROR MESSAGES

During the operation of the Program Compare Utility, error conditions are printed without interrupting multiple file processing. Other messages concerning Program Compare reports are shown above and are not error conditions.

MESSAGE	DESCRIPTION
ERROR: FILENAME (XYY) IS A DATA FILE.	This message indicates that one of the files to be compared is a data file. That compare operation is aborted. Check the displayed file name and disk address after processing ends to determine if the wrong file name was entered, or if the program to be compared was assigned a different file name.

CHAPTER 16
FORMAT 2260C DISK UTILITY PROGRAM

16.1 INTRODUCTION

The FORMAT 2260C DISK utility program formats the specified Model 2260C Disk Drive platter. Each disk platter must be formatted before it can be used to record data. Once formatted, a disk usually need never be reformatted.

The fixed and removable platters must be formatted individually. Whenever a new removable disk platter (disk cartridge) is obtained, the new disk cartridge must be formatted.

It is important to note that formatting destroys any data previously recorded on the disk formatted.

> NOTE:
>
> FORMAT 2260C DISK will only format a Model 2260C disk platter. Any disk address specified during Format 2260C disk operation must be a Model 2260C disk address; otherwise, spurious results and error conditions will occur.

16.2 OPERATING INSTRUCTIONS

DISPLAY	OPERATING INSTRUCTIONS
1.	1. From the ISS Utilities menu, load FORMAT 2260C DISK by touching the indicated Special Function Key.
2. ENTER DISK ADDRESS TO FORMAT?_	2. Enter the disk device address of the disk to be formatted, in the form xyy (e.g., 320, B20).

3.		3. The ISS disk(ette) may be removed at this time, if necessary, but should be remounted before touching RETURN in reply to step 6.
4.	DO YOU WANT TO FORMAT THIS PLATTER (Y/N)?_ FORMAT PLATTER xyy	4. The disk device address entered in step 2 is displayed (xyy). The prompt asks whether the user wants to format the platter at the disk address shown. Enter Y to begin formatting and skip to step 6. Enter N to enter a different disk address (return to step 2).
5.	FORMAT PLATTER /xyy FORMATTING...	5. No operator action is necessary. This message appears while formatting the disk, which takes about two minutes.
6.	KEY RETURN(EXEC) TO RESUME?_ FORMAT PLATTER /xyy FORMATTING COMPLETE	6. Upon successful completion of formatting, this prompt appears. Touch the RETURN key to obtain the ISS Utilities menu. If a different disk is present, a prompt appears requesting that the ISS disk be remounted.
7.	DO YOU WANT TO TRY AGAIN (Y/N)? FORMAT PLATTER /xyy FORMAT ERROR, RETURN CODE = nnnnnn	7. If a format error was encountered, this prompt appears. The RETURN CODE shown is described in the Disk Reference Manual. It is recommended that the operation be reattempted. To format this or a different platter, enter Y and return to step 2. To return to the ISS Utilities menu, enter N. If N is entered and the ISS disk has been replaced by a different diskette, a prompt requests that the ISS disk be remounted.

PART III

THE KEY FILE ACCESS METHOD

RELEASE 7

CHAPTER 17

GENERAL INFORMATION

17.1 INTRODUCTION TO DISK ACCESS METHODS

A disk access method provides a means of transferring data between memory and a direct access storage device such as disk or diskette. It enables records within a disk file to be rapidly located by certain conventions associated with the particular access method used.

Direct access (nonsequential) storage devices typically provide rapid access to randomly dispersed data on a disk(ette) platter by using a moveable read/write head for each platter. To fully utilize this desirable hardware feature, an access method is usually applied to certain data files (especially large data files) where rapid access to random records is required. The Key File Access Method (usually referred to as "KFAM") provides several means of rapid access to records within a file.

KFAM, although it is unique, resembles access methods usually categorized as "indexed sequential" or "indexed" access methods. A KFAM file consists of two files: (1) the file containing the data records, called a "User File" and (2) a file which contains an "index" for quickly locating specific User File records, called a "Key File". Within each data record in the User File is a "key", such as a social security number. Each key's value is unique within the same KFAM-7 file. The Key File contains system information used internally by KFAM, as well as the key and corresponding address of each record in the User (data) File which enables retrieval of records based on their keys. Key File information is automatically maintained by KFAM.

KFAM Release 7 (KFAM-7) provides the following features not always associated with "indexed sequential" access methods:

a. An entry in the Key File is maintained for each record in the User File, which allows records to be added in random order of their keys to the KFAM file and, thereafter, accessed by key or by key sequence. Record deletions by key are also provided.

b. Records entered in random order of their keys may be reordered by supplied KFAM-7 utility software which reorganizes User File records and corresponding Key File entries into ascending order of their keys. This provides efficient record access by ascending or descending key sequence.

c. Using supplied KFAM-7 utility programs, data files meeting KFAM-7 input record requirements may be converted to KFAM-7 User File format and a Key File created for the User File.

d. The Key File may be located on a different disk platter than the User File it indexes, thus minimizing disk hardware access times.

e. When the appropriate KFAM-7 marked subroutine is called which successfully opens (or closes) the requested User File, KFAM-7 automatically opens (or closes) the companion Key File.

f. KFAM-7 supports four file access modes, which collectively provide a controlled file access system for KFAM-7 disk files. Individual "stations" (user partitions) are granted or denied file access based on the requested access mode and access modes previously granted to other stations (user partitions) still accessing the file. Other multistation Security features are provided.

17.2 WHAT IS KFAM?

The 2200 BASIC-2 Language includes a group of statements used for disk operations known as the Automatic File Cataloging statements. Automatic File Cataloging statements create and maintain on a disk a catalog, or index, of the files stored on the disk. This catalog includes, among other things, the name given to each file and each file's starting and ending sector addresses.

Though the catalog system keeps track of where each file is located on a disk, and thereby allows files to be easily found, it does not keep track of the individual records within a file. For example, a given disk may have an employee file called "PAY", an accounts receivable file called "A/R", and an inventory file called "INVT". The disk catalog system keeps track of where each of these files is located. However, the "PAY" file may consist of 250 employee records, the "A/R" file of 400 customer records, and the "INVT" file of 5000 product records. KFAM is a system for keeping track of and locating these individual records within a file.

The Index of Data Records

For each file of records, KFAM creates and maintains an index within the Key File of the individual records and their locations within the User File. For the purpose of this index, each record is uniquely identified by some key field that marks it off from all other records. For example, for a payroll file, the employee name or social security number might be designated as the key field; for an inventory file a product number might be the key field. A record's key field is called its "key". The index constructed and maintained by KFAM can be thought of as a list of all the keys for a given file. Associated with each key in the index is the location of the record that the key identifies.

When a file is indexed by KFAM, one can say in a program, "Find me the record for product AB-4975-1." KFAM subroutines, accessed by the program, then search the Key File index and put the sector address of record AB-4975-1 into the User File's Current Sector Address parameter in the Device Table. (Refer to the Disk Reference Manual for information about Device Table characteristics.) The program simply executes a DATALOAD DC statement to read the desired record.

KFAM subroutines do all the work of searching and updating the Key File. There are KFAM subroutines to find records in a random key sequence and in ascending key sequence; there are subroutines to delete records, as well as find a location for a new record and add the new key to the Key File. Thus, the programmer who uses KFAM need never know how the Key File is constructed. KFAM subroutines carry out all the necessary operations on the Key File.

The Key File that KFAM constructs is a sophisticated tree structure, designed so that keys can be found quickly in a random key sequence, and even more quickly in ascending or descending key sequence. It allows keys to be added and deleted easily, without disturbing the organization of the Key File.

Whenever a KFAM subroutine is to find a record, or add a new key to the Key File and find a location for the new record in the User File, the KFAM subroutine puts the User File record location into the Current Sector Address parameter of the Device Table, opposite the file number being used for the User File. Thus, on return from the subroutine, an ordinary Catalog Mode DATALOAD DC or DATASAVE DC can be executed, and it will take place at the desired sector location. DA and BA access modes are also applicable to certain KFAM files.

The Evolution of KFAM-7

Prior to the current version of KFAM, known as KFAM-7, there were KFAM-5, KFAM-4, KFAM-3, KFAM-2, and the original version of the Key File Access Method, KFAM-1. Previous versions of KFAM were designed to operate on single-user, single-station 2200 Central Processors. Utility programs are provided to convert KFAM-3 files to KFAM-7 files and KFAM-4 files to KFAM-7 files. KFAM-5 files are media compatible with KFAM-7 and do not require conversion.

KFAM-7 is especially designed for the 2200MVP Central Processor. KFAM-7 subroutines--which can be accessed by all stations--are located in a global partition. Within this centralized global partition, KFAM-7 also maintains file access information by which it controls multistation access to shared KFAM-7 files. A maximum of 30 KFAM-7 files may be open on a multistation 2200MVP system without software modification; a maximum of eight KFAM-7 files may be open per station (see Section 20.3). KFAM-7 is only operable with a 2200MVP Central Processor.

The global partition, if it contains the entire set of KFAM-7 subroutines, occupies one 9K partition. The size of each user partition is determined during partition generation according to the memory requirements of the programs to be run in that partition. KFAM-7 utility programs require a user partition size of 9K.

If a user-written application program will be running in a user partition (station), between 1K and 2K is required for KFAM-7 variables within the user partition, depending upon the maximum number of KFAM-7 files to be open to that user partition (station).

Figure 17-1 shows a functional diagram of a typical KFAM-7 partition configuration. The shaded areas indicate memory required for KFAM-7 software; the unshaded areas indicate memory available for user-written application programs.

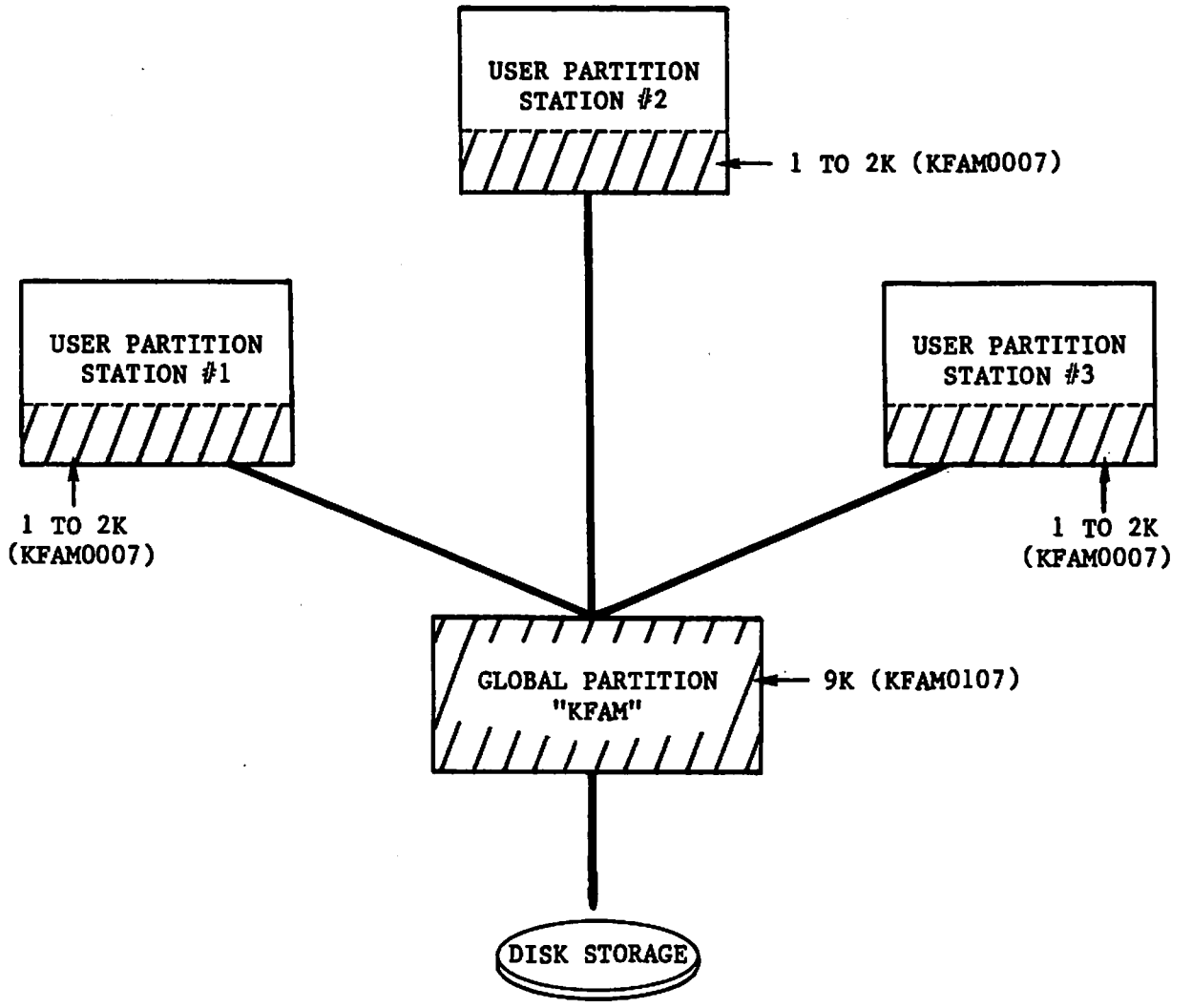


Figure 17-1. Functional Diagram of a Typical KFAM-7 Partition Configuration

17.3 KFAM FILE STRUCTURES

KFAM creates and maintains for each KFAM file certain information stored in its two components: the User File and the Key File. The User File contains mostly the data records which are maintained by the user. Other portions in the User File, however, contain specific information maintained by KFAM. In contrast, the entire contents of the Key File are maintained by KFAM in conjunction with the KFAM subroutines executed on the KFAM file. User Files must be data files (not program files).

User File Structure

The User File contains data records beginning with the first sector allocated to the User File. Data records are stored by the user within the User File via KFAM-7 subroutines and DATASAVE statements.

The end of live data is indicted by an "END" record which follows the last sector containing live data. The position of the "END" record is automatically controlled by KFAM-7 in conjunction with the KFAM-7 subroutines called by the user's program. In the next to last sector in the User File, KFAM maintains a "dummy" END record which, at the programmer's option, also contains recovery information necessary to recreate the Key File if it is accidentally destroyed. In the last sector allocated is the catalog trailer record. Contained within the catalog trailer record is that file's access table and password.

Access to the Key File is determined by access to the User File. The User File access table is examined before the User File is opened and determines whether User File access is granted or refused, based on the access mode requested and the access modes already granted to other stations. Once the User File is opened, the Key File is opened along with it, and some of the control information in the Key File (Key Directory Record) is loaded into global memory. The global memory control information handles all multiple station record access functions for any KFAM file open. Record access functions include completion codes, protected sectors, and other control information.

When a KFAM file is closed and when a FINDNEW, FINDNEW(HERE), or a DELETE subroutine is executed, the Key Directory Record is rewritten into the Key File.

Note that the END records and trailer record are indicated by the first two bytes of their respective sectors which contain specific HEX values. Figure 17-2 illustrates the User File structure layout.

The "END" record and the "dummy" END record's recovery information are rewritten only under certain conditions upon closing the file. The END record is rewritten when the file is closed, only if records have been added to the User File. Recovery information is written when the file is closed if record additions or deletions have occurred.

The presence of the END record following live data allows the file to be read sequentially, provided the user observes the following conventions:

1. All deleted records must be flagged with HEX(FF) in the first byte of the key.

2. If records are blocked (types A and C) every time a new sector is allocated (FINDNEW returns Q=1), all records in the block must be initialized with a HEX(FF) in the first byte of the key.

The END record following live data also allows the User File to be copied by the ISS Copy/Verify Utility, as discussed later in this manual. The position of the END record, of course, reflects the number of USED sectors and EXTRA (i.e., free) sectors in the file.

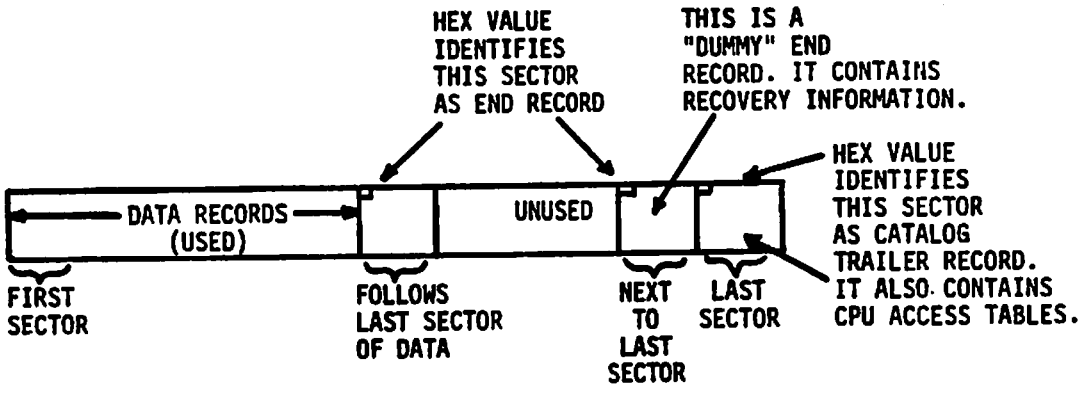


Figure 17-2. User File Structure

Key File Structure

The Key File is the means by which indexed record access occurs to randomly-dispersed records within the User File. The first sector of the Key File contains the Key Directory Record (KDR), and the remaining sectors contain Key Index Records (KIR's). The Key File also contains an END record to mark the end-of-live-data which is automatically rewritten as sectors are added to the Key File. The KDR is loaded into global memory following an Open and dynamically maintains information about each station's Current Sector Address, any sectors protected by a station, information about the KFAM file, and system (KFAM internal) information. The KDR is rewritten under certain conditions to reflect changes to its contents, for example, when records have been added or deleted. Recovery information, stored in the User File's dummy END record, consists of nearly all of the KDR.

The remaining sectors of the Key File contain Key Index Records (KIR's), which consist of Key Index Entries (KIE's). Key Index Entries are internal tables consisting of key values and corresponding record locations, by which KFAM provides indexed access to each User File record.

17.4 THE FUNCTIONAL COMPONENTS OF KFAM

KFAM-7 software components fall into the following categories:

1. Set-up Utilities: Standalone utility programs used to initialize a new KFAM File and to create a Key File for an already-existent User File (data file).

2. KFAM Subroutines: DEFFN' statement subroutines, contained within the global partition "KFAM", are used to open and close KFAM files, locate records in the User File, and add and delete keys from the Key File. KFAM subroutines are the operational heart of KFAM.

3. Supplementary Maintenance Utility Programs: Standalone routines, which perform a variety of tasks related to the maintenance of a Key File and User File, print the contents of a Key File and recover from certain operational accidents.

KFAM-7 is one of the software components available within the Integrated Support System, Release 3.7. Following ISS-3.7 start-up procedures, KFAM-7 utility programs may be chosen in reply to the KFAM-7 menu. ISS-3.7 start-up procedures maintain certain information in each station's file saved on disk(ette), including valid disk device addresses and the device address of a printer. This information is available to all ISS-3.7 utility programs and application programs through the use of common variables which, for instance, are used by KFAM-7 utilities to ensure that a disk address entered in reply to an operator-prompting message (prompt) is indeed a valid disk address.

17.5 HOW TO GET STARTED WITH KFAM

KFAM provides a means for accessing data records saved in a disk file. However, it does not process these User File records in any way. After it has found a record, the processing of the record, (loading it, updating it, saving it, etcetera) is left to the user-written program. Thus, to use KFAM one must have a working knowledge of elementary BASIC-2 and of the fundamentals of Catalog Mode disk operations.

It is strongly recommended that the first-time user of KFAM begin by setting up a dummy KFAM-7 file, and experiment with the subroutines and utilities on this file before attempting to operate on valuable files.

The following is a step-by-step outline of how to begin setting up KFAM files.

1. Read Chapter 18 which describes the five types of User File records which are acceptable to KFAM, limitations on the size and characteristics of the key field, and certain KFAM conventions which must be adhered to.

2. A Key File is stored as a cataloged file on a disk. It may reside on the same disk as the User File, or on another disk, (which must be mounted whenever the User File is accessed). A set-up utility program called INITIALIZE KFAM FILE must be run whenever a new KFAM file (Key File and User File) is to be established.

   INITIALIZE KFAM FILE calculates the required size of the Key File, given an estimate of the maximum number of records to be saved in the User File, and will catalog a Key File with the required number of sectors. It saves in the Key File some vital information about the User File, based on information supplied by the operator. It will also catalog a User File with the proper number of sectors if the User File does not already exist.

3. If a previously cataloged User File contains data, a second set-up utility program can be run after INITIALIZE KFAM FILE, called KEY FILE CREATION. The KEY FILE CREATION utility program reads a User File and creates an entry in the Key File for each record in the User File. Upon completion, KFAM subroutines may be used to provide access to records in the User File.

4. If the User File does not contain data, then after running INITIALIZE KFAM FILE use the KFAM subroutines in a user-written program to place data records in the User File and add the corresponding entries into the Key File simultaneously.

5. The KFAM subroutines are DEFFN' statement subroutines which perform standard tasks for files indexed by KFAM. In most cases, the KFAM-7 global module KFAM0107--which contains all KFAM-7 subroutines--is loaded and run in a 9K partition. Alternatively, the user may select the subroutines needed for the applications to be performed using the BUILD SUBROUTINE MODULE utility program, which creates a global

module containing the selected subroutines on the specified disk(ette). The module created by BUILD SUBROUTINE MODULE may be loaded into a partition instead of KFAM0107 if certain considerations are observed.

Subroutines are available to perform the following tasks.

TYPE AND NAME	FUNCTION
General Purpose Subroutines	
OPEN	Open specified User File and companion Key File.
CLOSE	Close User File and companion Key File.
RE-OPEN	Changes the access mode of a currently open KFAM File.
WRITE RECOVERY INFORMATION	Writes current file END record at end of active data in User file, and writes recovery information in the next-to-last sector (both of which would otherwise only occur when a file is closed) without closing the file.
Random Access Subroutine	
FINDOLD	Locate specified key in the Key File; set User File Current Sector Address to record in User File with that key.
Key Sequence Access Subroutines	
FINDFIRST	Locate record with lowest key in User File; set User File Current Sector Address to that sector.
FINDPREVIOUS	Locate previous record in User File in logical key sequence; set User File Current Sector Address to that sector. May be executed in any situation where FINDNEXT is allowed.
FINDNEXT	Locate next record in User File in logical key sequence; set User File Current Sector Address to that sector.
FINDLAST	Locate record with highest key in User File; set the User File Current Sector Address to that sector.

Add and Delete Subroutines

FINDNEW	Add specified key to Key File; allocate space for a new record in the User File, and set the User File Current Sector Address to that sector. Adds one to record count.
FINDNEW (HERE)	Add specified key to Key File; set the User File Current Sector Address to the sector where the new record is to be written. It is normally used to change the key of a deleted record; therefore, it is normally preceded by a DELETE. Adds one to the record count.
DELETE	Remove specified key from Key File; set the User File Current Sector Address to the record that has the deleted key. Subtracts one from the record count.

Special Purpose Subroutine

RELEASE	Allow a User File record, previously protected by one station, to be accessed by any station.

Since the KFAM subroutines allow records to be added and deleted, as well as changed, all regular file maintenance takes place in application programs which use KFAM subroutines. As a general rule all operations on the Key File are accomplished by the KFAM subroutines, while all operations on the User File are accomplished by user written statements in the application program.

6. Though the KFAM subroutines are the heart of the KFAM system, and perform most of the file maintenance, a group of Supplementary Maintenance Utilities are included to carry out certain maintenance tasks that will occasionally be required.

   a) The REORGANIZE Utilities: When a record is "deleted" by using the DELETE subroutine, its key and location are simply removed from the Key File. It then cannot be accessed by KFAM. However, the record itself in the User File is not removed. It is possible to reuse the spaces occupied by deleted records in the User File, but if this is not done the User File gradually becomes bloated with DELETED records. The REORGANIZE Utilities reorganize the User File by putting its records into ascending key sequence, eliminate DELETED records, and then automatically construct a new Key File for accessing the reorganized User File.

THE REORGANIZE SUBSYSTEM: is a standalone routine which reorganizes a file by outputting a new reorganized User File and Key File. The old Key File and User File are left intact. It is called by a short user-written set-up module which provides parameters for the reorganization. Also, it can be used to copy KFAM file.

REORGANIZE IN PLACE: is a utility program which reorganizes the User File and Key File in place. It should be used only for a file so large that adequate output files could not be mounted at the same time as the file to be reorganized.

b) The ISS COPY/VERIFY utility (if available) and the REALLOCATE KFAM FILE SPACE utility can be used together to copy a KFAM file and increase or decrease the amount of disk space allocated to the file. Use of REALLOCATE KFAM FILE SPACE is required after any KFAM file (User File and Key File) is copied by COPY/VERIFY. The REORGANIZE SUBSYSTEM may also be used to copy, change file space allocation, change the file name, as well as reorganize a KFAM File--all at the same time.

c) PRINT KEY FILE: This utility prints the complete contents of the access table and the current contents of the Key File with appropriate labeling of data. It can be useful as a diagnostic tool and helpful to advanced programmers who may wish to examine the Key File structure.

d) Recovery Utilities: A KEY FILE RECOVERY utility is provided to reconstruct a Key File in the event of its accidental destruction. The User File must be intact for this program to operate successfully.

Also available is a recovery utility called RESET ACCESS TABLE. KFAM-7 maintains certain information in various access tables about which stations are operating on the file. This access table information will contain erroneous information if a station fails to CLOSE a file it has opened, due to power failure or program error. The RESET ACCESS TABLE utility is provided to clear this erroneous information from the access tables.

e) The KFAM Conversion Utilities. Utility programs are provided to convert files from KFAM-3 to KFAM-7 and from KFAM-4 to KFAM-7. KFAM-5 files do not require conversion to KFAM-7 format.

17.6 KFAM-7 ACCESS MODES AND SECURITY FEATURES

KFAM-7, a modification of previous KFAM systems, is specifically designed for a 2200MVP multistation environment. It allows up to 16 stations to access a KFAM disk data file and includes protective procedures designed to prevent destructive intrusions of one station into the file operation of another station. These procedures are designed to offer protection consistent with the type of operation being performed, so that other stations can have safe maximum availability of the disk and the file.

There are four access modes available with KFAM-7. Certain conventions are to be followed for each access mode, as discussed below. The four access modes concern access to the User File; however, when the User File is opened for a station, the Key File is automatically opened along with it. The User File can be assigned a password. Reading and writing applies to both the User File and Key File in the descriptions below.

1. "Inquiry" is an access mode in which the station, once granted access (open), may only read, but other stations may read or write.

2. "Read Only" is an access mode in which the station, once granted access, may only read, and other stations may likewise only read.

3. "Shared" is an access mode in which a station, once granted access, may read or write, and other stations may likewise read or write.

4. "Exclusive" is a access mode whereby a station, once granted access, is the only station accessing that KFAM file until it is closed by that station.

In addition to the protective features provided by the various access modes, a sector protection option is provided for use in the Inquiry or Shared access modes, because (1) more than one station may be accessing a sector, and (2) this or another station may be writing (for instance, updating) that sector.

The protected sector, depending on the file type, protects (1) the entire block of records in the sector if blocked records are used, (2) the record itself if it is the only record in the sector, or (3) the entire record if the record is contained in more than one sector.

The record (sector) protection option is available to the programmer by an argument in the call to KFAM record access subroutines. Only that station can access a record, once protected, until that station executes another KFAM subroutine. Thus, for example, if a series of updates are being performed by a station, each access of a record turns off the protect flag set in global memory for the previous record accessed and optionally sets the protect flag for the next record.

A subroutine named RELEASE allows sector protection to be switched off, in case a substantial delay occurs before the next KFAM subroutine call by that station.

This discussion of record protection does not apply to the Read Only or Exclusive access modes, wherein the record protect argument is ignored by KFAM because either (1) only one station accesses the file at one time, or (2) the access mode does not allow writing, thus the contents of the KFAM file cannot be altered.

The user has control over disk hog mode use but rarely has any need for disk hog mode because of the available access modes and KFAM's built-in Key File integrity features. KFAM-7's Exclusive access mode is normally sufficient for updating records. However, for extremely critical updates involving multiple KFAM files, the user may activate and reactivate hog mode by means of a $OPEN statement. Disk hog mode is released for the User File, Key File, or both by KFAM-7 upon executing a subroutine or if the program execution is hogged by KFAM-7*, therefore the user program need not use the $CLOSE statement to release disk hog mode.

* KFAM-7 briefly hogs program execution during the following subroutines: OPEN, CLOSE, RE-OPEN, WRITE RECOVERY INFO., FINDNEW, FINDNEW(HERE), all multistation non-KFAM subroutines (OPEN, END, CLOSE), and during any subroutine occasionally when operating in the Shared or Exclusive access mode.

CHAPTER 18

KFAM REQUIREMENTS AND CONVENTIONS

18.1 THE USER FILE

An existing User File must be a cataloged disk data file, with or without an END record. It must be wholly contained on one disk platter. Five record types are supported. All records in a file must be the same type and of fixed length. The record types are:

Type "N" - No Blocking

Each record occupies one sector:

```
DIM A$25, B, B$40, C, C$40, D$40
DATASAVE DC #n, A$, B, B$, C, C$, D$
```

indicating one record per sector, with each record containing an A$, B, B$, C, C$, D$.

The key must be located in the same position within each record.

Records may be written in the "DC" mode, with control bytes, or in the "BA" mode, without control bytes.

Type "A" - Array Type Blocked Records

Records must be written in non-contiguous array form:

```
DIM A$(4)3, B(4), C$(4)20
DATASAVE DC #n, A$(), B(), C$()
```

indicating 4 records per block, each containing an A$, B, and C$. The block of records must be written using DATASAVE DC, that is, with control bytes; DATASAVE BA may not be used.

All records must have the same format. Control bytes must be included when calculating the record length.

The key must be located in the same position within each record. The key may be a part of a field, i.e., STR(C$, 11, 10), but may not span fields, may not include control bytes, and may not be a numeric field or any part of a numeric field.

The block of records may not exceed one sector in length.

There may not be more than 38 fields per record.

Type "C" - Contiguous Blocked Records

All records must be the same length.

All the fields of a given record are stored contiguously on the disk, for example:

```
DIM A1$3, C1$20, A2$3, C2$20, A3$3, C3$20, A4$3, C4$20
DATASAVE DC#n, A1$, B1, C1$, A2$, B2, C2$, A3$, B3, C3$,
A4$, B4, C4$
```

indicating 4 records per block, each containing an Aj$, Bj, and Cj$.

The key must be located in the same position within each record.

The block of records may not exceed one sector in length.

Records may be written in the "DC" mode, with control bytes, or in the "BA" mode without control bytes. However, if the file must be reorganized in place using the REORGANIZE IN PLACE utility, it must be written with control bytes. If written DC mode, control bytes must be included when calculating the record length.

Please note that type "C" records may not be acceptable to Wang disk sort software including SORT-4. KFAM-7 type "A" records are normally used instead.

Type "B" - BA Mode Blocked Records

Type "B" records resemble type "C" records in that both are contiguous blocked records, however, type "B" applies to records written in "BA" which, as you know, contain no control bytes. Type "B" records share the following characteristics with type "C" records:

a. The contiguous block of records must not exceed one sector in length.

b. All records must be the same length.

c. The key must be located in the same position in each record.

Because the type "B" records were written in the BA mode, they must be accessed in BA mode instead of DC mode, which is the way records are usually accessed after KFAM has located the requested record. Type "B" records must not be accessed by DATALOAD DC. Instead, the DATALOAD BA command is used and requires an absolute sector address.

Because absolute sector addressing is used, the starting address of the file must be obtained by a LIMITS statement inserted after the OPEN subroutine has been successfully executed, such as:

```
LIMITS T#U, N$, B, E, X
```

where: U = File Number, User File
N$= User File Name
B, E, X = variables set to receive User File starting sector, ending sector, and number of sectors used.

Upon return, the value B should be saved. When B is added to the record's relative sector address returned by KFAM, T6, this provides the absolute sector address necessary for the x value of the DATALOAD BA command shown in the example below:

```
DATALOAD BA T#U, (x,y) A$()
```

where: U = File Number User File
X = T6+B (absolute sector address, numeric form)

If KFAM returns an error condition (Q$ not blank), relative sector addresses T6 and T4$ are not defined. T4$ is simply the value of T6 in hexform; however, the value of B must be converted to hex form before adding T4$ and B, to produce a HEX value of x.

KFAM subroutines return a pointer, Q, which contains the number of the record within the block. To locate the starting byte within the 256-byte sector block, the following formula applies:

```
P = (Q-1)*L+1
```

where: P = starting byte within the block
L = record length
Q = pointer returned by KFAM

When the file is initially defined, using INITIALIZE KFAM FILE, the record specified must usually include one control byte per field within the record; however, with BA records ("B" type) there are no control bytes, thus the record length specified is the exact length of the records, with nothing added for control bytes.

Type "M" - Multiple Sector Records

Each record occupies more than one sector.

For example, the following is a two-sector record:

```
DIM D$(6)64
DATASAVE DC #n, D$()
```

Each record occupies the same number of sectors.

The key must be located in the same position within each record. The key may be located in any sector of the record, but may not span sectors.

Records may be written in the "DC" mode, with control bytes, or in the "BA" mode, without control bytes.

Records may be up to 255 sectors in length. However, the following restrictions apply in REORGANIZE IN PLACE:

a. Records may not exceed 40 sectors in length.

b. Reorganization cannot be executed in 9K of memory if the record length exceeds 8 sectors.

User File Name

The User File name, as recorded in the disk catalog, must conform to the following conventions:

The 5th character must be the letter "F".

The 6th character must be a digit 0-9.

18.2 THE KEY

Each record's key in the User File may be from 2 to 30 bytes of alphanumeric data (including hexadecimal data or packed numbers). The key must not be a numeric field.

The first byte of an active key may not contain the value HEX(FF). The value HEX(FF) in the first byte of a key in the User File indicates that the record has been deleted from the Key File.

The key may not contain a value of all bytes HEX(00). (This corresponds to the packed number 0, or the binary number 0, as a key value.) This lowest possible value is reserved for the system.

Duplicate keys are not allowed. User File records need not be arranged in ascending order of their keys within the file. KFAM automatically creates the (indexing) Key File in whatever order the keys are within the User File.

18.3 THE KEY FILE

The Key File name is constructed by INITIALIZE KFAM FILE from the User File name, as follows:

The 5th character in the User File name is changed from "F" to "K".

The 6th character is assigned the Key File number. This is always 1 unless multiple Key Files are maintained for the one User File, in which case it may be any digit 1-9.

Size of the Key File

The first sector of the Key File contains the Key Descriptor Record (KDR). The KDR contains control information necessary for KFAM.

The remaining sectors of the Key File are available for Key Index Records (KIR's). Each KIR occupies one sector and contains Key Index Entries (KIE's). The KIE is a field containing a key and a 3-byte pointer. The key is the same as one of the keys in the User File. The pointer points to a User record on the disk, either directly or indirectly. The maximum number of KIE's per KIR is given by:

```
N = INT (240/(K+3))

where:  K = Key length
        3 = pointer length
        N = maximum KIE's per KIR
```

The average number, A, of KIE's per KIR is calculated (conservatively) as follows:

```
A = INT(N*.6)
```

The number of sectors, required for the Key File, for a given number of records, R, is as follows:

```
S = INT(R/(A-1))+5
```

18.4 DEVICE ADDRESSES

Valid ISS disk device addresses and the ISS printer address are specified during ISS start-up operation. Valid disk addresses include the address specified as the ISS loading address (system disk).

When copying or creating a KFAM file it is recommended that the Key File and User File be stored contiguously if on the same disk, or that the Key File be stored on a different disk than the User File. This will minimize disk read/write head access times (track-to-track access), thus improving throughput characteristics. Procedures for using non-KFAM subroutines to catalog multiple files on a disk are described in Section 28.4.

18.5 COPYING KFAM FILES

Back-up copies of disks containing KFAM files should be made regularly using the COPY statement. Use of the MOVE statement destroys the file's access table. For file copy operations, however, where file space allocations can be changed, the procedures below must be carefully followed.

During INITIALIZE KFAM FILE operation, the END sector is originally written for the User File and Key File, and is automatically rewritten thereafter by KFAM when the file is closed to reflect changes.

However, when copying a KFAM User File and Key File, the ISS Utility COPY/VERIFY is required for both files. One of the parameters requested by COPY/VERIFY is a value of EXTRA SECTORS for the file(s), which is directly related to the END sector's position and USED sectors.

For example, suppose a KFAM User File has an allocation of 1000 sectors. If the file is copied and has 253 sectors USED, then the EXTRA SECTORS value for COPY/VERIFY must be 747 sectors to maintain the allocated space of 1000 sectors. Similarly, if records are then added so that 264 sectors are used, then 736 EXTRA SECTORS must be entered. The same procedure for User File EXTRA SECTORS also applies to the Key File, whose USED sectors value increases automatically under the control of KFAM as new keys (User File records) are added to a file. KFAM does not, however, change allocated file space as required.

It is extremely important to note that with KFAM files the internal system information maintained in the KDR is <u>not</u> updated by COPY/VERIFY to reflect changed file allocations.

Thus after any KFAM User File or Key File is copied by COPY/VERIFY, the REALLOCATE KFAM FILE SPACE Utility Program must be run to update this critical system information, for each file copied.

In summary, to copy a KFAM File, the following procedures are required for both the User File and Key File.

1. Use a LIST DC statement or equivalent to obtain the values of sectors USED, and sectors ALLOCATED for each file.

2. For each file, calculate the EXTRA SECTORS by subtracting sectors USED from sectors ALLOCATED.

3. For each User File and Key File, run the COPY/VERIFY Utility in the PART mode (PART allows different EXTRA SECTOR values for a series of files), and specify (1) input file name, (2) output file name, and (3) extra sectors assigned to the output file, for each file.

4. Upon completion of COPY/VERIFY operations, run REALLOCATE KFAM FILE SPACE for each file copied to the COPY/VERIFY output device address.

5. The copied files are now ready for access by other software.

CHAPTER 19

THE KFAM SET-UP UTILITY PROGRAMS

19.1 OVERVIEW OF INITIALIZE KFAM FILE

INITIALIZE KFAM FILE must be run as the first step in setting up any KFAM file, whether the file is, or is not, cataloged. If the file already contains data records, INITIALIZE KFAM FILE should be followed by KEY FILE CREATION (see Figure 19-1).

INITIALIZE KFAM FILE optionally catalogs an area on disk for the User File, the Key File, or both, or it operates with an existing User File, Key File, or both. It sets up the Key Directory Record (KDR), the first record of the Key File, containing vital information about the User File and the Key File, based on information supplied by the operator. It then creates a "null" (empty) Key File. The KDR occupies the first sector, and is followed by a Key Index Record (KIR), in the "null" Key File's second sector. An "END" record is written following the first two sectors.

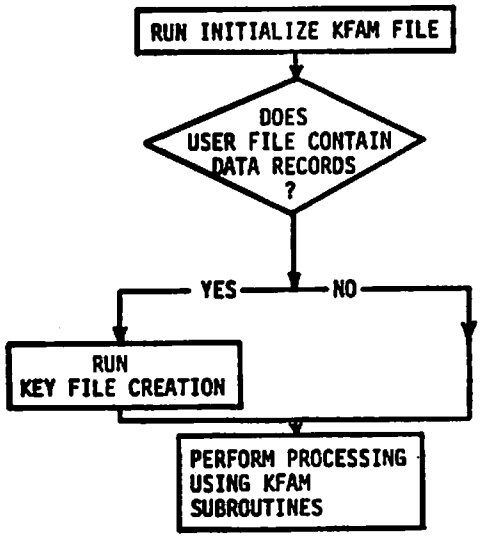


Figure 19-1. KFAM Set-up Utilities

In the User File, the position of the END record depends on whether the User File was previously cataloged or not. With a previously uncataloged file just created, the END record is written in the first sector. With a previously cataloged file, the position of the END record, if any, remains unchanged. In both cases, recovery information (KDR) is stored in the next-to-last sector (dummy END record) of the User File, and the access table and password is stored in the last sector (hardware trailer) of the User File. The access table is cleared upon being written.

Information Required by INITIALIZE KFAM FILE

INITIALIZE KFAM FILE requires that the following information be supplied by the operator:

```
User File Name
Device Address for User File:  any valid ISS disk device address
Password (if any)
Is User File Cataloged?:  Y OR N
Key File Number:  1-9
Device Address for Key File:  any valid ISS device address
Is Key File Cataloged?:  Y OR N
Record Type:  A, B, C, M, or N
Logical Record Length (Type A, B, or C):  nnn
Blocking Factor (Type A, B, or C):  nn
Sectors per Record (Type M):  nnn
Key Length:  2-30
Starting Position of Key:  nnnnn
Estimated Number of Records:  nnnnn
Are File Specifications OK?:  Y OR N
Hard Copy Printout?:  Y OR N
```

This is a formidable set of questions for an operator. It is suggested that this utility be run by an application programmer, or that a programmer write a set of specific answers for a specific KFAM file, as a supplement to the general operating instructions below.

Some of the answers to the above questions are not obvious and require some discussion:

User File Name

The User File Name must conform to the KFAM naming convention. The 5th byte must be "F". The 6th byte must be a digit 0-9. The remaining bytes may be any alphanumeric characters. The User File may be an active or scratched cataloged file; if the User File is not a cataloged file, INITIALIZE KFAM FILE will create the User File.

Password

Entry of a Password, which allows access to the User File, is required only if the User File is already cataloged and active, and had a Password previously assigned to it. When creating a new User File, the entry of the Password assigns that Password to the User File, thus requiring entry of that Password whenever any station seeks to access that file. If no characters are entered (i.e., key RETURN), a Password of blanks is assigned to the new User File. With an already cataloged User File, a reply of RETURN indicates a Password of blanks was previously assigned to this file and thus entry of a Password is not required.

Key File Number

Normally, 1 should be entered. However, if multiple Key Files are to be used to index the same User File, they must be uniquely identified by the Key File Number, which can be any digit from 1 to 9. Multiple Key Files per User File are not supported by Wang Laboratories, Inc.

The Key File name is derived from the User File name, by replacing the "F" in position 5 with "K", and the digit in position 6 with the Key File Number. The Key File may be active, scratched, or not cataloged.

Record Type

KFAM supports five different record types:

Type A: Array type blocked records.

More than one data record is contained in a sector. The block of records is written as an array (non-contiguously) within a sector, e.g.,

```
DIM A$(4)12, B(4), C$(4)36
DATASAVE DC #n, A$(), B()  C$()
```

indicating 4 records per sector, each record containing an A$, B, and C$.

Type C: Contiguous blocked records.

More than one data record is contained in a sector. Each record occupies a contiguous amount of space on the disk. For example, there are three records per sector, each containing a key (K$) and data (D$):

```
DIM K1$12, D1$64, K2$12, D2$64, K3$12, D3$64
DATASAVE DC #n, K1$, D1$, K2$, D3$, K3$, D3$
```

Type N: No blocking

Each data record occupies one sector.

Type M: Multiple Sectors per record.

Each data record occupies two or more sectors.

Type B: BA Mode Blocked Records

More than one data record is contained within a sector. The blocked records occupy a contiguous amount of space. These records resemble Type C records, except that Type B records are written and accessed in BA Mode instead of DC Mode, and thus contain no control bytes.

Logical Record Length

The logical record length for record types A, B, and C is calculated as follows:

a. Add up the lengths of the fields contained in a single record (numeric fields are 8 bytes long).

b. For record types A and C (DC mode) only, add 1 per field of the record, for control bytes. Control bytes must be included when calculating the record length, with the exception of type C records written in BA mode.

For example, in the above example for type A records, the record length is 59. In the above example for type C records, the record length is 78.

All records of the file must have the same length. For type A, all records must also have the same format, for example, a 12-byte alpha field, followed by a numeric field, followed by a 36-byte alpha field, each field contained in an array of 4 elements.

Blocking Factor

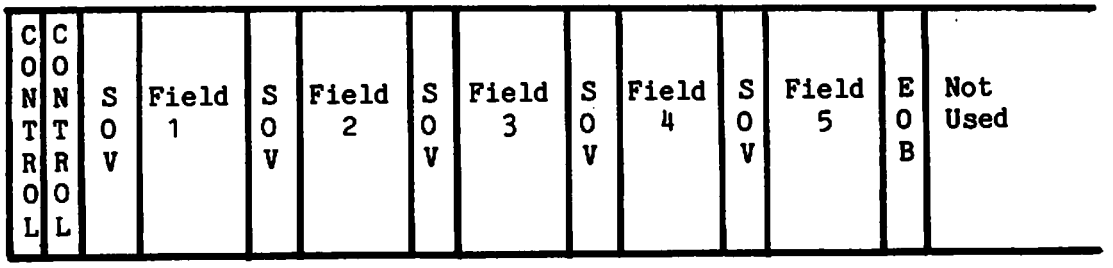
The blocking factor is the number of records per sector (Type A, B, and C).

Starting Position of Key

This is the absolute starting position of the key within the sector or sectors, except for type A records, where it is the position within the record plus two for sector control bytes. This requires some explanation.

When a record is written on disk, in the normal mode (DATASAVE DC or DATASAVE DA), two control bytes are written at the start of the sector. Following these two control bytes, the Start-of-Value (SOV) control byte for the first data value is written, followed by the data itself. Then the SOV control byte for the second value and the second value itself are written, and so on. An end-of-block (EOB) control byte is written following the last data value written.

The layout of the sector on disk looks like this:



For the purposes of determining the starting position of the key, the bytes of the sector are numbered from 0 to 255. The starting control bytes are bytes 0 and 1. The SOV control byte for the first field is byte 2. The first byte of data is byte 3. The second field starts in byte 3 + L1 + 1, where L1 is the length of the first field, and so on.

The starting position of the key is the number of the first byte of the key. For blocked records, the blocking is ignored when calculating the starting position of the key. It is calculated as if there were only one record per block. The KFAM utilities make the necessary adjustments to calculate the position of the key in subsequent records within the sector.

In particular, with type A records, the blocking should be ignored when calculating the starting position of the key. Given the record in the example:

```
DIM A$(4)12, B(4), C$(4)36
DATASAVE DC #n, A$(), B(), C$()
```

the actual layout of the sector is as follows:

Bytes	Contents
0,1	Control bytes
2	SOV
3-14	A$(1)
15	SOV
16-27	A$(2)
28	SOV
29-40	A$(3)
41	SOV
42-53	A$(4)
54	SOV
55-62	B(1)
63	SOV
64-71	B(2)
72	SOV
73-80	B(3)

Bytes	Contents
81	SOV
82-89	B(4)
90	SOV
91-126	C$(1)
127	SOV
128-163	C$(2)
164	SOV
165-200	C$(3)
201	SOV
202-237	C$(4)
238	EOB

The fact that there are four records per sector should be ignored in determining the starting position of the key. The sector should be seen as if it contained only one record, as follows:

Bytes	Contents
0,1	Control bytes
2	SOV
3-14	A$(1)
15	SOV
16-23	B(1)
24	SOV
25-60	C$(1)
61	EOB

If the key starts in the first byte of C$(), the starting key position is 25, and not 91, as would be indicated by the actual blocking.

For record types C, M and N, it is possible to have records written in the DATASAVE BA mode. In that case, no control bytes are inserted, and the starting position of the key is exactly where it is located in the array defining the record (starting byte 0).

For record type B, it is necessary to write records in BA mode, and the starting position of the key is exactly as described in the preceding paragraph.

For record type M, it is possible to have the key begin in the second, or higher, sector of the record. In that case, add 256 for each sector preceding the one containing the key, and then add the starting position of the key within the sector (first byte of the sector = 0).

When writing multiple sectors in the normal mode (DATASAVE DC or DATASAVE DA), it is necessary to determine which field will begin the second sector, etc. The 2200 System does not write partial fields in a sector. Where there is not room to write the next field in the current sector, the 2200 System writes an EOB control byte, leaves the rest of the space unused, and starts another sector. For example, if the record is defined:

```
DIM D$(6)64
DATASAVE DC #n, D$()
```

the record occupies 2 sectors, as follows:

Bytes	Contents
First sector:	
0,1	Control bytes
2	SOV
3-66	D$(1)
67	SOV
68-131	D$(2)
132	SOV
133-196	D$(3)
197	EOB
198-255	Not used (58 bytes, not room to write next complete field)
Second sector:	
0,1	Control bytes
2	SOV
3-66	D$(4)
67	SOV
68-131	D$(5)
132	SOV
133-196	D$(6)
197	EOB
198-255	Not used (end of data)

Once the actual record layout is determined, then the starting position of the key can be calculated. If the key occupies, for example, the first 8 bytes of D$(4), then the starting position of the key is 259, and not 198 as might be calculated by ignoring the actual way that the system writes records.

Estimated Number of Records

This is the maximum number of records that the User File will contain. If the User File is not yet cataloged, the program will catalog enough sectors to hold this many records. If the User File is already cataloged, the program checks that enough space is cataloged to contain this many records.

The program also calculates the size of the Key File, based on the estimated number of records. If the Key File is not yet cataloged, the program catalogs the required number of sectors. If the Key File is already cataloged, the program checks to see that enough space is cataloged. If there is not enough space cataloged, the program issues a warning message.

The estimated number of records should be calculated in advance, as the maximum number of records which the User File will contain, plus an estimate of the number of deleted records which will be in the file when it is at its maximum size.

This estimate is not critical. It can be revised later, using the ISS COPY/VERIFY Utility, followed by REALLOCATE KFAM SPACE.

Hard Copy Printout

Output goes to the ISS start-up printer address assigned to this station. For example, if printer address = 215, output will go to the printer with a device address of 215. If the printer address = 000, output will appear on the screen.

19.2 INITIALIZE KFAM FILE OPERATING INSTRUCTIONS

	DISPLAY		INSTRUCTIONS
1.		1.	From the KFAM-7 menu, load the INITIALIZE KFAM FILE utility by touching the specified Special Function Key. Mount the disk platter(s) containing, or to contain, the User File and Key File.
2.	ENTER USER FILE NAME (SSSSFJNN)	2.	Enter the name of the User File. (The word "enter" includes touching the RETURN key after visually verifying the entry made.)
			NOTE: Error messages and recovery procedures appear in Appendix A. Also, although not shown here, dashes appear with the prompts on the actual displays to allow easy entry of parameters.
3.	ENTER USER FILE DEVICE ADDRESS	3.	Enter the xyy form of the User File disk device address.

4. ENTER PASSWORD	4. With an already cataloged and active User File, an entry to this prompt is only required if the cataloged User File has a Password assigned other than blanks to it; if no Password is assigned, key RETURN without entering any characters. With a non-cataloged User File or a scratched User File, any entry made here will be assigned to the User File as a Password, which (later) will be required in order for any station to access this KFAM File. If a Password need not be assigned to this User File, key RETURN without entering any characters to assign a Password of blanks.
5. IS USER FILE CATALOGED? (Y OR N)	5. Enter "Y" if the User File already exists as an active or scratched data file. Enter "N" if the User File does not exist.
6. ENTER KEY FILE NUMBER (NORMAL = 1)	6. Normally enter 1. If there is more than one Key File for a single User File, the Key File number is used to distinguish the Key Files. The Key File Number can be any digit from 1 to 9.
7. ENTER KEY FILE DEVICE ADDRESS	7. Enter the xyy form of the Key File device address.
8. IS KEY FILE CATALOGED? (Y OR N)	8. Enter "Y" if space has already been cataloged for a Key File. Enter "N" if the Key File has not been cataloged.

9.	ENTER RECORD TYPE (A,B,C,N,M)	9. Enter A, B, C, N or M. A=array type blocking B=contiguous BA mode blocking C=contiguous blocking N=no blocking M=multiple sector records If record type A, B, or C, proceed with Step 10, below. If record type M, proceed with Step 12, below. If record type N, proceed with Step 13, below.
10.	ENTER LOGICAL RECORD LENGTH	10. Enter logical record length. See above for calculation of logical record length.
11.	ENTER BLOCKING FACTOR	11. Enter number of records per sector. Proceed with Step 13, below.
12.	ENTER NUMBER OF SECTORS PER RECORD	12. Enter the number of sectors per record. Record Type M only.
13.	ENTER KEY LENGTH	13. Enter key length (2 to 30). All record types.
14.	ENTER STARTING POSITION OF KEY	14. Enter starting position of key field within sector. See above for calculation of starting position of key.
15.	ENTER ESTIMATED NUMBER OF RECORDS	15. Enter estimated maximum number of records in User File. See above for calculation of number of records.
16.		16. The system calculates disk space required for User File and Key File.
17.		17. The system displays file specifications on the screen.

18.	ARE FILE SPECIFICATIONS OK (Y OR N)	18. Check file specifications displayed on the screen. Enter Y to continue. If erroneous file specifications are displayed, enter N and return to the KFAM menu.
19.	DO YOU WANT A HARD COPY PRINTOUT OF FILE DESCRIPTION? (Y OR N)?	19. For a hard copy printout, mount paper on printer and enter Y. Otherwise, enter N and skip to Step 21.
20.		20. The system prints file specifications on the printer by hogging the printer, if Y was the reply to Step 19. This is an exact duplicate of the screen display, Step 17.
21.		21. The system initializes the User File and Key File. NOTE: If WAITING FOR PRINTER appears, correct the printer, which must be ON and SELECTED, or wait until it is available. The program will continue automatically.
22.		22. The KFAM menu appears upon completion.

19.3 KEY FILE CREATION UTILITY

The KEY FILE CREATION utility creates a Key File for the data records in an existing User File. INITIALIZE KFAM FILE must have been run first, to initialize both the User File and the Key File.

KEY FILE CREATION ignores any records which have HEX(FF) in the first byte of the key, under the assumption that they are deleted records. It also ignores records which have duplicate keys, but prints the relative sector number and record number where such duplicate keys are encountered.

The utility optionally allows the operator to enter the key for the last record in the User File in physical sequence (see step 7 below). The program can use this to detect the end-of-file condition. The value of the last key should be made available to the operator before running this program, unless the User File has a valid END record.

The KFAM file is opened in the Exclusive mode.

KEY FILE CREATION Operating Instructions

	DISPLAY		INSTRUCTIONS
1.		1.	From the KFAM-7 menu, load the KEY FILE CREATION utility by touching the specified Special Function Key. The disks containing the User File and Key File must be on-line.
2.	ENTER USER FILE NAME (SSSSFJNN)	2.	Enter the name of the User File. NOTE: Error messages and recovery procedures appear in Appendix A. Although not shown here, prompting dashes accompany the actual prompts.
3.	ENTER USER FILE DEVICE ADDRESS	3.	Enter the xyy form of the User File disk device address.
4.	ENTER PASSWORD	4.	An entry to this prompt is only required if the User File has a Password assigned to it, in which case the Password must be entered exactly as previously to this User File. If a Password was not assigned, key RETURN without entering any characters.
5.	ENTER KEY FILE NUMBER (NORMAL=1)	5.	Enter the Key File Number. The Key File Number should always be 1, unless there are multiple Key Files for a single User File, in which case the Key File Number may be any digit from 1 to 9. In any case it must have been initialized.

6.	ENTER KEY FILE DEVICE ADDRESS	6. Enter the xyy form of the Key File Device address.
7.	ENTER LAST KEY	7. The last key need not be entered if the User File has an END record following live data, and all records to be deleted, including inactive blocked records in the last used sector, have been flagged by a HEX (FF) in the first byte of the key. In this case, key RETURN without entering any characters; otherwise, enter the key of the last record in the User File.
8.	(Record locations and keys are displayed on the screen so that the operator can check the progress of the program.)	8. The system opens the files, sets up the screen display, and creates the Key File. Execution messages are printed or displayed on lines 7-10 respectively, if the ISS start-up printer device address indicates a printer (e.g., 215) or is 000. With displayed output, key CONTINUE, then RETURN to continue if an error message is encountered (see Appendix A).
9.		9. The KFAM-7 menu appears upon completion.

CHAPTER 20

KFAM-7 SUBROUTINES AND BUILD SUBROUTINE MODULE

20.1 OVERVIEW OF KFAM-7 SUBROUTINES

The KFAM-7 subroutines are designed to simplify the file access and maintenance operations most frequently performed on files organized by KFAM-7. Included are DEFFN' statement subroutines to add new records to a file, delete existing records, locate existing records, and access a file's records in ascending or descending key sequence.

A single KFAM-7 file consists of two cataloged disk files, a User File and its Key File. KFAM subroutines never alter the data in the User File. They operate upon the data in the Key File, locate a record, and update the Key File whenever a record is to be added or deleted. Their function in relation to the User File is only to set the User File's Current Sector Address to the location of the desired record in the User File and, for blocked records, to pass back to the application program the record location within the sector. This process is initiated when the application program passes a key to the KFAM subroutine in one of the GOSUB' statement arguments. Upon completion of the KFAM subroutine, the application program must perform the proper operation on the User File.

An internal mechanism in global memory called a "queue" ensures for all stations operating in the Inquiry and Shared access modes attempting to access the same file that subroutine execution is performed on a first-come-first-served basis for the respective file. Similarly, KFAM-7 takes the precaution of "hogging" program execution during certain subroutines to protect the integrity of the control information contained within global memory.

Just as KFAM-7 does not operate upon the User File, so the application program should never operate directly upon the Key File. All operations involving the Key File, including OPEN and CLOSE, should be accomplished via the KFAM-7 subroutines. The functions performed by the KFAM-7 subroutines are:

Name	Function
OPEN	Open specified User File and companion Key File, which previously were created by the INITILIZE KFAM FILE utility program.
DELETE	Remove specified key from Key File; set this station's Current Sector Address to the record in the User File whose key has been deleted from the Key File. Subtracts one from record count.
FINDOLD	Locate specified key in the Key File; set this station's Current Sector Address to the record in the User File with that key.
FINDNEW	Add specified key to Key File; allocate space for a new record in the User File, and set this station's Current Sector Address to the location for the new User File record. Adds one to record count.
FINDNEW (HERE)	Add specified key to Key File; set this station's Current Sector Address to User File sector where the new record is to be written. Adds one to record count; normally follows a DELETE (changes deleted record's key).
FINDFIRST	Locate record with lowest key in User File; set this station's Current Sector Address to that User File sector.
FINDLAST	Locate record with highest key in User File; set this station's Current Sector Address to that User File sector.
FINDNEXT	Locate next record in User File in logical key sequence; set this station's Current Sector Address to that User File sector.

FINDPREVIOUS	Locate previous record in User File in logical key sequence; set this station's Current Sector Address to that User File sector. May be executed in any situation where FINDNEXT is allowed.
RELEASE	Allows a User File sector, previously protected by a station, to be accessed by other stations.
RE-OPEN	Changes the access mode of a currently open User File.
WRITE RECOVERY INFORMATION	Writes current file END record at end of active data in User File, and writes recovery information in the next-to-last sector--which would otherwise only occur when a file is closed--without closing the file.
CLOSE	Close User File and companion Key File.

Reserved Variables

KFAM-7 subroutines use Q, R, T, and V variables and arrays (alpha and numeric) for storage of critical internal pointers. The user-written program should, therefore, strictly avoid the use of any Q, R, T, or V variables. Also, note that @Q, @T, and @V global variables are reserved for KFAM-7 use.

Reserved DEFFN' Statement Subroutines

With the presence of a global partition containing the KFAM-7 DEFFN' statement subroutines, the need to reserve the DEFFN' statements provided by KFAM-7 for exclusive KFAM-7 use is extremely important. When a GOSUB' statement is encountered, the calling user partition is searched for the corresponding DEFFN' statement before the currently selected global memory partition is searched for the corresponding DEFFN' statement. The global program text contained in module "KFAM0107" includes DEFFN'212 through '219 and DEFFN'230 through '239 (inclusive). The user's application should avoid use of these reserved DEFFN' statements. Similarly, any other global partition should not be named "KFAM" (e.g., DEFFN' @PART "KFAM").

Identification of KFAM Files

A User File and Key File can be thought of collectively as a KFAM file. To use a KFAM file, the User File and the Key File must be open simultaneously. Thus, each KFAM file requires two "slots" or "rows" in the station's Device Table. With 2200MVP systems, there are sixteen Device Table "slots" per station numbered #0 to #15, which allow up to eight KFAM files to

be open. Before opening any KFAM-7 file, the file numbers associated with the User File and Key File must be identified to their disk addresses using a SELECT statement, e.g., SELECT #3/B10, #4/310.

The OPEN subroutine must be used to open a Key File and User File, before any KFAM file access operations can take place. The user-written application program must pass to the OPEN subroutine the file numbers to be used for the Key File and User File. The application program also passes to OPEN a digit, which will be used to identify this KFAM file (User File/Key File) for all other KFAM subroutines. This digit, 1-8, is called the "KFAM ID number." When passed to the OPEN subroutine, OPEN establishes this number as the single identifier for the pair of cataloged files being opened. In subsequent operations, while these files are open, they are identified collectively simply by passing the KFAM I.D. Number to the desired KFAM subroutine.

The file number for the User File is employed only when a record is to be written to or read from a User File. Then, the file number of the User File (#0 - #15) must be specified in the DATASAVE DC or DATALOAD DC statement. The KFAM I.D. Number is not used. In general, the file number of the Key File is never used, since any reference to the Key File should be via a KFAM subroutine. Note that accidental use of the Key File's file number in place of the User File's file number in a DATASAVE DC statement causes the user data to be written over data in the Key File. This destroys the Key File.

It is possible to have the same User File open concurrently with more than one Key File, but each such pair (User File/Key File) must have a different KFAM I.D. Number, and the User File must be referenced by a different file number in each case. Multiple Key Files per User File is allowed but is not supported.

Key File Recovery Information

The KFAM maintenance utilities include a program called KEY FILE RECOVERY. The purpose of this program is to reconstruct a Key File from an existing User File, in the event that the Key File is accidentally destroyed. Unlike the program KEY FILE CREATION, it does not require that the key of the last physical record in the User File be known. However, it does require that all user-written application programs operating on the file adhere to two conventions:

1. All DELETED records in the User File must have hex (FF) as the first byte of the key. This means that after a program calls the DELETE subroutine, it must then save hex (FF) into the first byte of the record's key. Also, when blocked records are added to a KFAM file, if the new sector is only partially filled with added records and "inactive" records exist within that sector, the inactive records must contain hex FF in the first byte of the key.

2. With KFAM-7, CLOSE is a system requirement. If BUILD SUBROUTINE MODULE was used and the CLOSE WITH RECOVERY INFO option was not selected, the user must include the WRITE RECOVERY INFO subroutine and must execute the WRITE RECOVERY INFO subroutine before closing a KFAM file whenever FINDNEW has been used.

20.2 KFAM ACCESS MODES

There are four KFAM file access modes designed for efficient multistation, shared file operation. The chosen access mode can be changed once file access is gained by calling the RE-OPEN subroutine. Each access mode is listed in Table 20-1 with a description of record access conventions to be followed, as well as which file access modes are allowed and are not allowed by other stations attempting to access the file. Under the column RECORD ACCESS DESCRIPTION in Table 20-1, "read" and "write" applies to both the User File and Key File. It is the user's responsibility not to update a file opened in the Inquiry or Read-Only access modes.

Table 20-1. KFAM Multistation Access Modes

ACCESS MODE	RECORD ACCESS DESCRIPTION	DOES ALLOW (OPEN SUCCESSFUL)	DOES NOT ALLOW (ACCESS CONFLICT)
Inquiry (1)	Indicates this station will read only, but other stations may read or write. User File sector protection available. No updating allowed.	Inquiry, Read Only Shared	Exclusive
Read Only (2)	Indicates this station will read only, and other stations may read only. User File sector protection does not apply. No updating allowed.	Inquiry, Read Only	Shared, Exclusive
Shared (3)	Indicates this station may read or write, and other stations may read or write. User File sector protection available.	Inquiry, Shared	Read Only, Exclusive
Exclusive (4)	Indicates that only this station has the file open. Other stations cannot access this file. User File sector protection does not apply.	None	Inquiry, Read Only, Shared, Exclusive

Inquiry and Shared Access Modes

Inquiry and Shared access modes assume that another station may be updating records and/or changing the Key File structure. Record protection is available for use. Record protection is indicated by a protect flag argument and is available for the following record access subroutines: FINDOLD, DELETE, FINDNEW, FINDNEW(HERE), FINDFIRST, FINDLAST, FINDNEXT, FINDPREVIOUS.

> NOTE:
>
> The following subroutines require files they access to be Opened in Shared or Exclusive mode: DELETE, FINDNEW, FINDNEW(HERE).

Other Access Modes

In the Read Only and Exclusive access modes, the Key File cannot be changed by another station because (1) in the Read Only mode only reading is allowed and (2) in the Exclusive mode only one station can access the KFAM file. Record protection of the User File is ignored. This saves processing time over the Inquiry or Shared modes and is made possible by the protection of the KFAM file built into these access modes.

KFAM Utility Access Modes

Should it be necessary to run an application program simultaneously with other application programs or with KFAM utility programs, some planning of multistation file use may prove helpful. The following KFAM utility programs access KFAM files in the Exclusive mode:

KEY FILE CREATION Utility Program
REALLOCATE KFAM FILE SPACE Utility Program
REORGANIZE IN PLACE Utility Program
REORGANIZE SUBSYSTEM Standalone Routine
CONVERT KFAM-3 TO KFAM-7 Utility Program
CONVERT KFAM-4 TO KFAM-7 Utility Program
RESET ACCESS TABLE Utility Program
KEY FILE RECOVERY Utility Program

PRINT KEY FILE uses the Read-Only access mode. BUILD SUBROUTINE MODULE hogs the disk during its execution.

20.3 PROCEDURES FOR PROGRAMMING WITH KFAM-7

In addition to the calling sequences, access mode selection, and the previously mentioned conventions necessary for Key File recovery and identification of KFAM files, the following list of considerations should be incorporated into KFAM-7 application programs.

1. Each application program must contain the KFAM-7 variables from the module "KFAM0007". The array elements of the arrays on line 225 may be changed to any value from 1 through 8 to allow more than 3 KFAM files to be open at any one time (preset to 3). These KFAM-7 variables require about 1,000 bytes plus 87 bytes for each KFAM file to be open. Also, if the user needs more than 30 files open per 2200MVP System, see Section 26.2, "Internal Storage: Global Memory".

   It is recommended that the user load KFAM0007 from the KFAM-7 diskette and edit line 225 as required before entering the program text. The statement lines originating from KFAM0007 should be saved along with the user's program text and thus are incorporated into the user's application program. The programmer should observe the BASIC-2 Language conventions applicable to COM and DIM statements and may renumber the KFAM0007 statement lines if required.

2. Before opening any KFAM-7 file, the variable S2 must be equated to the station number (partition number) currently in use. Station numbers may range from 1 through 16 and may be equated either by the program or by an operator entry during ISS start-up.

3. Before calling any KFAM-7 subroutine, the application program must execute the following program statement within the program and any associated overlays.

   ```
   SELECT @PART "KFAM"
   ```

   This program statement selects the global partition "KFAM" for this user partition. If any other SELECT @PART statement is executed afterwards within the user's application program, the SELECT @PART "KFAM" statement must be re-executed to reselect the global partition "KFAM" before any KFAM-7 subroutines are called (only one global partition may be selected by each partition).

4. The $OPEN and $CLOSE statements are available to respectively activate and deactivate hogging of a particular peripheral such as a disk drive or a printer connected to the CPU. For example, because a printer connected to the CPU may be accessible to multiple partitions, the printout from one partition may be interspersed with another partition's printout. To avoid this situation, a partition should hog the printer ($OPEN) until printing has been completed, and then release the printer hog ($CLOSE). Similarly, an application program may take advantage of the 2200MVP's programmable

interrupt feature to determine when an currently hogged printer (or any other peripheral) is available and take advantage of its availability.

5. The application program should provide an error recovery procedure that will close all KFAM-7 files open or provide the operator with a means of closing files. The ERROR or SELECT ERROR statements may be used in conjunction with the ERR function to provide access to error recovery routines which might be self-correcting and thus avoid the need to close KFAM-7 files (especially for "background" user partitions).

6. In the Inquiry or Shared access modes, the record protect option should be implemented on a record if a DATASAVE will be executed on the record following return from that subroutine. Updating records, adding new records, and flagging deleted records all require that a DATASAVE be executed; therefore, the protect flag should be set for all these operations. Also, if a DATALOAD is first executed to read a record to be updated, a DBACKSPACE statement is needed before executing the DATASAVE because a DATALOAD statement automatically increments the Current Sector Address. Operations which only execute DATALOAD on the record should not set record protection.

7. The use of DATASAVE DC END to write the END record (following the last sector of live data records) must <u>not</u> be performed. KFAM maintains the END record automatically in the Key File's KDR and updates the END record in the User File when the file is closed. A DATASAVE DC END would destroy the file, as would any MOVE statement.

8. The application program, when accessing User File records, must check for a return code of Q$ = "B", indicating that the record sought is protected. On a Q$ = "B" condition, the application program should execute the $BREAK statement before reexecuting the subroutine. For example,

```
4230    GO TO 4250
4240    $BREAK 5 :REM WAIT FIVE "TURNS"
4250    GOSUB' 237 (2,1):REM FINDNEXT
4260    IF Q$ = "B" THEN 4240: REM IF BUSY, WAIT AND RETRY
```

9. In general, a file should not be opened in Exclusive mode, except in the following circumstances:

   a) A time-related operation must take place with the file status fixed as of the beginning of the operation. For example, printing a report at the end of an accounting period.

   b) If maximum file access speed is needed. When a file is open in Exclusive mode, the KFAM-7 subroutines can search the Key File without concern for protected records.

   c) As an alternative to disk hog mode use.

10. Application programs must never write trailer records of any kind into the User File. In general, application programs must never make any assumptions about the status of User File sectors other than those specifically returned by a subroutine. For example, under KFAM-7 it is possible for the next sequential record location, after that returned by a FINDNEW, to be already occupied by a live record written by another station.

11. If the previous subroutine call set the protect flag and there may be a long delay before the next KFAM-7 subroutine call on that file, application programs should execute the RELEASE subroutine.

    One might consider any keyboard entry operation as a long delay and execute RELEASE prior to the keyboard entry. Alternatively, a Special Function Key subroutine (i.e., use the RETURN statement after the DEFFN' statement) that executes RELEASE may be made available during all keyboard entry operations. The operator would then be instructed to touch the specified Special Function Key if there is any delay prior to responding.

12. The CLOSE subroutine must be executed at the conclusion of operations on any KFAM-7 file. The operator should always have available a procedure for CLOSING the file in the event of program malfunctions or other disaster. (If the CPU's power is turned off without CLOSING the file, the access table retains a notation for a "phantom" station; the RESET ACCESS TABLE utility must be run.) A Special Function Key subroutine might be made available to CLOSE a file at any time.

13. Perhaps the most frequently encountered programming error is the one that is most often taken for granted, that is, to check the return code contained in alpha-variable Q$ after each KFAM subroutine call. As shown in Figure 20-1, after calling any KFAM subroutine, the value of Q$ must be checked upon return. If Q$ does not return a blank, the appropriate error routine should be accessed. Again, a Q$="B" on a record access subroutine indicates a protected (busy) sector, and a delayed retry is recommended. For other values of Q$, line numbers or other means of isolating the probable cause might be displayed.

    Also, the value of numeric variable Q indicates the record number in the sector if blocked records are used, or Q equals one if unblocked records are used. The value of Q need only be checked with blocked records.

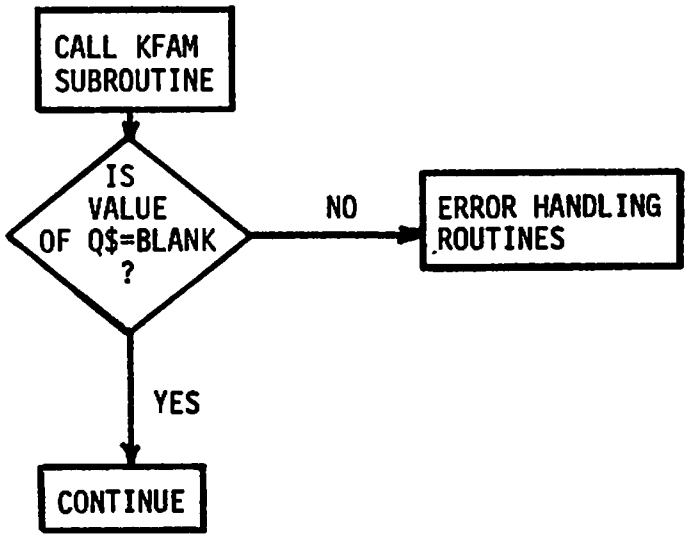


Figure 20-1. KFAM Subroutine Return Code Logic

20.4 BUILD SUBROUTINE MODULE UTILITY PROGRAM

BUILD SUBROUTINE MODULE creates a module at the specified disk address containing the KFAM subroutines chosen by the programmer. It allows the programmer to include in the module to be loaded into global memory only those subroutines and subroutine capabilities actually needed and thereby minimizes the amount of global memory required for KFAM-7 subroutines. This is an alternative to using the KFAM-7 module KFAM0107 which contains all KFAM-7 subroutines. With KFAM0107, the CLOSE WITH RECOVERY INFO is executed when the CLOSE subroutine is called. When a module is created by BUILD SUBROUTINE MODULE, one of the BUILD SUBROUTINE MODULE options determines whether the CLOSE is to be executed with or without RECOVERY INFO.

> NOTE:
> KFAM-7 utilities require the following subroutines to be contained within the global partition "KFAM": OPEN, FINDOLD, FINDNEW (HERE), FIRSTFIND, FINDNEXT, and CLOSE WITH RECOVERY INFORMATION. Also, there must be only one global partition "KFAM" per 2200MVP system at any one time.

BUILD SUBROUTINE MODULE Options

The following subroutine options appear in step 4 of the Operating Instructions below; their descriptions appear below in Table 20-2. Note that S.F. Keys 29, 30, and 31 are discussed in the Operating Instructions following Table 20-2.

Table 20-2. BUILD SUBROUTINE MODULE Options

S.F. KEY/SUBROUTINE	DESCRIPTION
Selections 01 to 10	KFAM subroutines
11/239 CLOSE	KFAM Close subroutine which does not write recovery information necessary for KEY FILE RECOVERY. Usually only chosen if records will not be added or deleted.
12/CLOSE WITH RECOVERY INFO	KFAM Close subroutine which writes recovery information (KDR) in User File necessary for KEY FILE RECOVERY.
Selections 13 and 14	KFAM subroutines
15/217 MUX OPEN	For non-KFAM files (sequential files), this subroutine opens a file if it is cataloged.
16/ MUX OPEN NEW	For non-KFAM files (sequential files), this subroutine opens and catalogs (creates) the file.
17/218 MUX END	For non-KFAM files, this subroutine writes an END record.
18/219 MUX CLOSE	For non-KFAM files, this subroutine closes the file.

Operating Instructions - BUILD SUBROUTINE MODULE

	DISPLAY	INSTRUCTIONS
1.		1. From the KFAM-7 menu, load BUILD SUBROUTINE MODULE by touching the specified Special Function Key.
2.	ENTER THE NAME OF PROGRAM TO BE GENERATED?	2. Enter the name of the program file which is to contain the selected subroutines, maximum of 8 characters. If a file of the same name is not already cataloged, the utility allocates just enough space for the selected subroutines. If a file of the same name is already cataloged, that file is used for the output program and is overwritten. If the file is a data file, it is changed to a program file. If extra space is desired in the output file, it should be cataloged in advance as a data file. NOTE: Error messages appear in Appendix A. Although not shown here, prompting dashes accompany the displayed prompts.
3.	ENTER OUTPUT PROGRAM DEVICE ADDRESS	3. Enter the disk device address at which this subroutine module (program file) will be stored on.
4.	KEY THE DESIRED FUNCTIONS (All options are displayed. The file name and number of files entered in steps 2 and 4 are also displayed.)	4. Refer to Table 20-2 above for the subroutines required for this module. Touch the appropriate Special Function Key for the option required; an asterisk will appear to the left of each selection.

		To cancel the options selected, touch S.F. Key 29 and return to step 4 above. To process those options indicated by asterisks (verify these first), touch S.F. Key 30 and go to step 6. To abort this operation and obtain the KFAM menu, touch S.F. Key 31.
5.	PHASE 2 - BUILDING MODULE (file name entered in Step 2. appears)	5. The output module is generated. The KFAM menu appears.

20.5 CALLING THE KFAM-7 SUBROUTINES

The DEFFN' statement which marks each KFAM subroutine in global memory requires certain parameter values to be passed from the GOSUB' statement which calls it. Parameter values passed are assigned to certain variables within the subroutine. The parameters (arguments) required are denoted in this manual as "symbolic variables" (i.e., "dummy variables") following the appropriate GOSUB' statement. Symbolic variables are not the actual variables required in an argument list. Instead, symbolic variables indicate whether a numeric or an alphanumeric expression is required in place of the symbolic variable.

If a symbolic variable's name is numeric, a numeric expression such as a number or a user-defined numeric variable or array is required in its place. If a symbolic variable's name is alphanumeric, an alphanumeric expression such as an alphanumeric literal (in quotes) or a user-defined alphanumeric variable or array is required in its place. This convention attempts to ensure that an alphanumeric expression (argument) is not assigned to a numeric variable in a subroutine, and vice versa.

Generally, the name chosen for a symbolic variable is the first letter of the associated parameter's name. In the actual program, the programmer may use any value or expression valid for use in a GOSUB' statement. Zeros in the general statement represent parameters which are not used by KFAM; they should be included as zeros in the GOSUB' statement.

For example, the general statement:

```
GOSUB' 233 (I,P,A$,0)
```

may be written as:

```
GOSUB'233(I,P,K$,0)
GOSUB'233 (2,1,"A48-3029",0)
GOSUB'233(F1+1,0,STR(P1$,7,8),0)
etc.
```

The symbolic variable names for KFAM-7 and their meanings are described in Table 20-3.

Table 20-3. KFAM Subroutine Argument Symbolic Variables

Symbolic Variable	Meaning
I	KFAM I.D. Number (1-8)
K	File number assigned to the Key File
U	File number assigned to the User File
F	Key File number (1-9), specified as the 6th character in the Key File name, as assigned in INITIALIZE KFAM FILE.
A$	The record key (alphanumeric).
N$	User File name.
A	Access Mode as described in Table 5-1: 1 - Inquiry 2 - Read Only 3 - Shared 4 - Exclusive
P$	File Password (if any) assigned this file during INITIALIZE KFAM FILE.
P	Protect flag. If P=0 or P=2 do not protect this record. If P=1 or P=3 protect this record.
xxx	Device address, Key File
yyy	Device address, User File

Return Codes Q and Q$

Upon returning to the main line program from the subroutines, the variables Q and Q$ contain the following information:

Q returns the record position indicator for blocked files (i.e., files with more than one record per sector). The record position indicator is a numeric value which specifies the position of a desired record within a block. For example, if Q=2, the key passed to the subroutine specifies the second record in the block. For unblocked records, Q is returned as 1 and may be ignored.

Q is not defined following the OPEN, WRITE RECOVERY INFORMATION, RELEASE, RE-OPEN, or CLOSE subroutines.

Q$ contains the completion return code. It indicates the result of the particular operation. The possible values of Q$ and their meanings are described in Table 20-4.

Table 20-4. KFAM Subroutine Q$ Return Codes

Q$ Value	Meaning
blank	The subroutine execution was successful.
D	Duplicate key (attempting to add a duplicate key to the file). The Key File is unchanged. For OPEN User File not found, OPEN issued to open file, or other file disposition conflict.
E	End of file (FINDNEXT, FINDPREVIOUS only).
N	Key not found. Also "null" File.
S	No more space, either for the User File or the Key File, or 8 levels of index have been exhausted attempting to add a record to the file. The Key File is unchanged. (FINDNEW and FINDNEW(HERE) only.)
	For an OPEN, this indicates too many files open to accommodate opening this file (global memory tables are full).
B	Busy Signal. The User File record or block of records being accessed has been "protected" by another station.
A	Access Mode conflict (OPEN only). See Table 20-1.
P	Invalid Password.
X	Improper call to a KFAM subroutine, (argument values erroneous, etc.).

If Q$ is anything other than blank, the User File Current Sector Address parameter is undefined, and the value of Q is undefined.

Immediately upon return from any of the subroutines, the user's application program should check Q$ for possible error indications. For instance, if Q$<>" " indicates an error.

The system assumes there are no programming errors in the user's program. The KFAM subroutines can perform improperly and destroy a file if the parameters supplied by the application program are erroneous. Therefore, during the testing stage, it is recommended that the user keep a backup file so that test data can be recovered if it is destroyed.

The subroutines check data errors and the kinds of errors likely to occur during normal operation such as duplicate key, key not found, or no more space. Errors resulting in Q$ = "X", ERR P37, or other ERR codes may occur if the global subroutine module, as generated in BUILD SUBROUTINE MODULE, does not contain all subroutines referenced by the user program. The following errors, which are programming errors, may or may not be caught by the subroutines:

Error	Q$ Value or ERR Code
KFAM I.D. Number not an integer between 1 and 8.	X ERR P34
KFAM I.D. Number is the same as I.D. Number for a file already open.	X
File to be opened is already open.	X
Individual file numbers not integers between 0 and 15.	ERR P57 ERR P34
Individual file number is duplicate of another file number.	X
File name not in proper format, with 5th byte="F" and 6th byte a 0 (zero).	ERR D82 ERR D84
Key File number not an integer from 1 to 9.	ERR X72
File to be accessed has not been opened.	X
File names are not correct or do not exist on the disk platters specified.	ERR D82 ERR D84

20.6 OPEN (DEFFN'230)

The OPEN subroutine is used to open a User File and its companion Key File which were previously created (initialized) by INITIALIZE KFAM FILE. OPEN must be executed prior to execution of any other KFAM subroutine. In the OPEN subroutine, a pair of modified multistation non-KFAM OPEN subroutines are executed to open the named User File and its companion Key File. A SELECT statement must be provided to associate each file number with its corresponding disk device address. OPEN assigns a specified KFAM I.D. Number to the pair of files. To call the OPEN subroutine, the following statements are necessary within the user's program (note the presence of symbolic variables):

```
S2 = station number (1-16)
SELECT #K/xxx, #U/yyy
GOSUB' 230 (I,K,U,F,N$,A,P$, "xxx", "yyy")
```

For The SELECT Statement

S2 is the station number (partition number). #K is the Key File number, and xxx is the Key File disk address. #U is the User File number, and yyy is the User File disk address (not used).

"#U" is the file number to be associated with the User File. "#U" must be used in all subsequent DATASAVE or DATALOAD statements to reference the User File.

For The GOSUB' Statement

"I" is the KFAM I.D. Number which is to be associated with the newly opened file, and must be used to reference the file in subsequent KFAM subroutines. "I" can be a number from 1 to 8.

"K" is the file number to be assigned to the Key File (see #K above).

"U" is the file number to be assigned to the User File (see #U above).

"F" is the Key File number (the sixth character in the Key File name, it may be an integer from 1 to 9, but normally it is 1).

"N$" is the name of the User File to be opened. The Key File name need not be specified; it is built from the User File name and the Key File number by KFAM itself.

"A" is the Access Mode as described in Table 20-1:

1 - Inquiry
2 - Read Only
3 - Shared
4 - Exclusive

"P$" is the File Password assigned this file during INITIALIZE KFAM FILE.

"xxx" is the device address, Key File.

"yyy" is the device address, User File (although not used by KFAM-7, an argument value must be provided in the GOSUB' 230 statement).

Return Codes for OPEN

Q$ = " " (space) if the subroutine execution was successful.

Q$=X	Station number, S2, not 1-16
X	File already opened
X	Duplicate file #
X	Access mode invalid
X	Key file not found
D	Named User File is program file or scratched or not found
P	Invalid password
A	Access mode conflict (see Table 20-1)
S	Internal global partition of open files is full. See Chapter 26, global array @T$(), for additional information.
ERR P34	KFAM I.D. out of bounds
ERR P57	File # not 0-15
ERR X71	Key File Number not 0-9 (0 is invalid but not checked)

Changing Access Modes

To change a KFAM-7 file's access mode, the station may either call the RE-OPEN subroutine or CLOSE the file and OPEN it with a different access mode.

20.7 DELETE (DEFFN' 231)

The DELETE subroutine deletes from the Key File a specified key and its associated record location pointer. The Current Sector Address for the User File is set to the location of the record whose key has been deleted, and for blocked records the variable Q indicates the record within the sector. The record itself in the User File is not altered or removed. Thus, although the record is not physically removed from the User File, its key entry is removed from the Key File, and the record can no longer be accessed through KFAM. Shared or Exclusive access is required. This subroutine's execution subtracts one record from the record count.

The calling sequence for DELETE is:

GOSUB' 231 (I, P, A$)

"I" is the KFAM I.D. Number assigned to the file in an OPEN subroutine.

"P" is the record protect option. P equal to 1 or 3 indicates record protection; P equal to 0 or 2 indicates no record protection.

"A$" is the key of the record that is to be deleted from the file.

DELETE Return Codes

Q$ = "B" Busy Signal. The record sought is protected by another station.
Q$ = "N" if the key passed cannot be found in the Key File.
Q$ = "X" for file not open, invalid key containing HEX(FF), or access mode not Shared or Exclusive.
Q$ = " " ("space") if the subroutine executed properly.

After calling a DELETE subroutine and checking for its successful completion, the application program should flag the DELETED record in the User File by changing the first byte of the deleted record's key to hex (FF). Note the use of the DBACKSPACE statement in statement line 4100; this statement is necessary to access the record just deleted because the DATALOAD DC statement's execution causes the Current Sector Address to be automatically incremented by one sector. For unblocked files this can be done as follows:

Suppose:

```
DIM A$15, H(4,4), J(6)
```

and

```
DATA SAVE DC #1, A$, H(), J()
```

define a type "N" record where A$ is the key field.

The DELETE and flag operation might look like this:

```
4060 GOSUB' 231 (1, 1, A$): REM DELETE
4065 IF Q$ = "B" THEN 4060:REM BUSY TRY AGAIN
4070 IF Q$<>" " THEN 6000:REM UNSUCCESSFUL
4080 DATA LOAD DC #1, A$, H(), J()
4090 STR(A$,1,1)=HEX(FF):REM HEX(FF) IN 1ST BYTE OF KEY
4100 DBACKSPACE #1,1S:REM RECORDS ARE 1 SECTOR LONG
4110 DATA SAVE DC #1,A$,H(),J()
.
.
.
6000 STOP "DELETE UNSUCCESSFUL"
```

The space occupied by DELETED records in the User File can be immediately reused; this normally requires special techniques together with the use of FINDNEW(HERE). For information on these techniques, see Section 27.2.

20.8 FINDOLD (DEFFN' 232)

The FINDOLD subroutine is used to locate a desired record in the User File. Following subroutine execution, the Current Sector Address for the User File is set to the sector address of the record whose key was passed. For blocked records, variable Q indicates the record within the sector. The record can then be read with a DATALOAD statement. The calling sequence is:

GOSUB' 232 (I, P, A$)

"I" is the KFAM I.D. Number assigned to the file in the OPEN subroutine.

"P" is the record protect option. P equal to 1 or 3 indicates record protection; P equal to 0 or 2 indicates no record protection.

"A$" is the key of the record being sought.

FINDOLD Return Codes

Q$ = "B" Busy Signal. The record sought is protected by another station.
Q$ = "N" if the specified key is not located in the Key File.
Q$ = "X" for file not open.
Q$ = " " ("space") if the key was located successfully.

20.9 FINDNEW (DEFFN' 233)

The FINDNEW subroutine is used to enter a new key in the Key File and to find a location for the new record in the User File. FINDNEW enters the key (argument) into the Key File and then sets the Current Sector Address for the User File to an available User File location for writing a new record. For blocked records, variable Q indicates the record within the sector. After calling FINDNEW, the normal procedure is to test Q$, test Q if records are blocked, and execute a DATASAVE statement. Shared or Exclusive access is required. This subroutine adds one record to the record count.

GOSUB' 233 (I,P,A$,0)

"I" is the KFAM I.D. Number assigned to the file in an OPEN subroutine.

"P" is the record protect option. P equal to 1 or 3 indicates record protection; P equal to 0 or 2 indicates no record protection.

"A$" is the new key to be entered in the Key File.

FINDNEW Return Codes

Q$ = "B" Busy Signal. The record (or block) sought is protected by another station.
Q$ = "D" if the key specified is a duplicate of one already in the Key File.
Q$ = "S" if there is no space in the User File for another record, or in the Key File for another key entry, or 8 index levels have been exhausted.
Q$ = "X" if file not open, invalid key of HEX(FF) or zeros, or if access mode is not Shared or Exclusive.
Q$ = " " ("space") if the key was entered without difficulty.

NOTE:

The User File location returned by FINDNEW is unoccupied by live data, but is not necessarily at the end of all live data in the User File.

Key File sectors are normally split 50/50 or a "bias" of .5. KFAM sets the bias at .5 when a file is opened. An experienced programmer may set the bias following the Open, for subsequent FINDNEW or FINDNEW (HERE) operations. The array V8 (I) may be set to any value between .2 and .8 (I = KFAM file I.D.). Records entered in sequential order are packed best at .2, in random order at .5, and in backwards sequence at .8. Also see Section 26.2.

The following example illustrates the procedure for adding a record to type A blocked files following FINDNEW. Note the test on Q before the DATASAVE, and that the protect flag is set by FINDNEW.

```
4100        INPUT "KEY FIELD", A$     :REM OPERATOR ENTERS KEY
4120        GOSUB '233 (1,1,A$,0)     :REM FINDNEW
4130    REM TEST COMPLETION CODE
4135        IF Q$ = "B" THEN 4120     :REM BUSY TRY AGAIN
4140        IF Q$ = "D" THEN 5010     :REM DUPLICATE KEY?
4150        IF Q$ = "S" THEN 5050     :REM FILE FULL?
4160        IF Q$ <>" " THEN 5060     :REM ERROR?
4170    REM NEW BLOCK OR OLD?
4180        IF Q = 1 THEN 4220  :REM FIRST RECORD IN NEW BLOCK?
4185    REM READ EXISTING RECORDS IN BLOCK
4190        DATA LOAD DC #2, A$(), B$(), C(), D()
4200        DBACKSPACE #2, 1S: REM BACKSPACE AFTER DATA LOAD
4210    REM ASSIGN RECORD VALUES TO PROPER ARRAY ELEMENTS
4220        A$(Q) = A$
4230        INPUT "SECOND FIELD", B$(Q)
4240        INPUT "THIRD FIELD", C(Q)
4250        INPUT "FOURTH FIELD", D(Q)
4260    REM SAVE BLOCK IN USER FILE
4270        DATA SAVE DC #2, A$(),B$(),C(),D()
5000    REM ERROR ROUTINES
5010        STOP "KEY ALREADY IN KEY FILE"
5050        STOP "KEY FILE OR USER FILE IS FULL"
5060        STOP "FINDNEW ERROR"
```

20.10 FINDNEW(HERE) (DEFFN' 234)

The FINDNEW(HERE) subroutine is a special purpose subroutine which can be implemented to reuse the User File space occupied by DELETED records or to change the value of the key of an existing record. It adds a new key to the Key File. Unlike FINDNEW, however, the User File location, which it associates with that key, is the User File location returned by the last KFAM subroutine call which must be DELETE. To use FINDNEW(HERE) to reuse the User

File space occupied by DELETED records, see Section 27.2. An illustration of the use of FINDNEW(HERE) to change the value of the key of an existing record is shown below. Shared or Exclusive access is required. This subroutine adds one to the record count.

The calling sequence is:

```
GOSUB' 234 (I,P,A$,0)
```

The FINDNEW(HERE) argument list is identical to the argument list for FINDNEW (see FINDNEW).

FINDNEW (HERE) Return Codes

Q$ = "B" Busy Signal. The record or block sought is protected by another station.
Q$ = "X" if file not open, invalid key HEX(FF), if pointer (T4$) is out of bounds, or file not opened with Shared or Exclusive access.
Q$ = "D" if the key specified is a duplicate of a key already in the Key File.
Q$ = "S" if there is no space in the Key File for another entry or if 8 index levels have been exhausted.
Q$ = " " (space) if the subroutine executed properly.

The following example illustrates the use of FINDNEW (HERE) following DELETE:

```
5000 GOSUB '231 (1,0,"ABCD") :REM DELETE "ABCD" FROM KEY FILE
5005 IF Q$ = "B" THEN 5000:REM BUSY
5010 IF Q$ = "X" THEN 5130
5040 IF Q$ = "N" THEN 5150
5050 GOSUB '234 (1,1,"EFGH",0) :REM SET "PROTECT", INSERT "EFGH" IN KEY
FILE
5060 IF Q$ = "X" THEN 5140
5070 IF Q$ = "D" THEN 5160
5075 IF Q$="S" THEN 5170
5080 DATALOAD DC #2,A$,B$,C$,N
5090 A$ = "EFGH" :REM CHANGE KEY TO "EFGH"
5100 DBACKSPACE #2, 1S
5110 DATASAVE DC #2,A$,B$,C$,N
5115 GOSUB'239(1) :REM CLOSE FILES
5120 END
5130 STOP "ERROR IN 'DELETE' CALLING SEQUENCE"
5140 STOP "ERROR IN 'FINDNEW(HERE)' CALLING SEQUENCE"
5150 STOP "KEY NOT FOUND"
5160 STOP "DUPLICATE KEY"
5170 STOP "NO SPACE"
```

Key file sectors are normally split 50/50. The array V8() is retained so that bias may be set by the experienced system programmer. V8(I), where I is the KFAM file I.D. number, is set to .5 when the file is opened. Following the Open, an experienced programmer may set it to any value in the range .2 to .8. Records entered in sequential order are packed best at .2, in random order at .5, and in backwards sequence at .8. Also see Section 26.2.

20.11 FINDFIRST (DEFFN' 235)

The FINDFIRST subroutine sets the Current Sector Address for the User File to the first record in logical key sequence. For blocked records, variable Q indicates the record within the sector. A DATALOAD statement can be used after FINDFIRST to read the record. In order to access a KFAM file in ascending key sequence, FINDFIRST is called followed by FINDNEXT calls until the end of file is reached (FINDNEXT return code Q$ = "E"). The calling sequence is:

```
GOSUB' 235 (I,P)
```

"I" is the KFAM I.D. Number assigned to the file in an OPEN subroutine.

"P" is the record protect option. P equal to 1 or 3 indicates record protection; P equal to 0 or 2 indicates no record protection.

FINDFIRST Return Codes

Q$ = "B" Busy Signal. The record sought is protected by another station.
Q$ = "N" if the User File contains no records.
Q$ = "X" if file not open.
Q$ = " " (space) if the subroutine executed properly.

20.12 FINDLAST (DEFFN' 236)

The FINDLAST subroutine sets the Current Sector Address for the User File to the last record in logical key sequence. For blocked records, the variable Q is set to the record position within the sector. A DATALOAD statement can be executed following FINDLAST to read the record. In order to access a KFAM file in descending key sequence, FINDLAST is called followed by FINDPREVIOUS calls until the beginning of the file is reached (FINDPREVIOUS return code Q$ = "E"). The calling sequence is:

```
GOSUB' 236 (I,P)
```

"I" is the KFAM I.D. Number assigned to the file in an OPEN subroutine.

"P" is the record protect option. P equal to 1 or 3 indicates record protection; P equal to 0 or 2 indicates no record protection.

FINDLAST Return Codes

Q$ = "B" Busy Signal. The record sought is protected by another station.
Q$ = "N" for a null file.
Q$ = "X" if file not open.
Q$ = " " (space) if the subroutine executes normally.

20.13 FINDNEXT (DEFFN' 237)

The FINDNEXT subroutine sets the Current Sector Address for the User File to the record immediately following (in logical key sequence) the last record accessed by KFAM. For blocked records, the variable Q indicates the record within the sector. A DATALOAD statement is executed following FINDNEXT to read the record. FINDNEXT is useful for processing files in ascending key sequence. The calling statement is:

```
GOSUB' 237 (I,P)
```

"I" is the KFAM I.D. Number assigned to the file in an OPEN subroutine.

"P" is the record protect option. P equal to 1 or 3 indicates record protection; P equal to 0 or 2 indicates no record protection.

FINDNEXT Return Codes

Q$ = "B" busy signal. The record sought is protected by another station. A retry is recommended following a $BREAK statement. FINDNEXT goes back to the original key, so that the same record that caused the busy signal is accessed on a retry.
Q$ = "X" if file not open or if next record not defined because previous operation returned an error condition (Q$ not blank). Exception: FINDOLD returning Q$ = "N" may be followed by FINDNEXT.
Q$ = "E" if the previous reference was to the last record in logical key sequence (end of file).

Otherwise, Q$ = " " (space).

NOTE:

FINDNEXT cannot be executed as the first subroutine following an OPEN subroutine call (use FINDFIRST). Also, FINDNEXT cannot normally be executed immediately following any subroutine which returned Q$ = "X" or "E". Otherwise, FINDNEXT will locate the next sequential key, following any subroutine.

If FINDNEXT is executed after a FINDNEXT which returned Q$ = "B" (the Busy Signal), it will attempt to access the same record that it previously found to be protected.

If FINDNEXT is executed following a FINDOLD that returned Q$ = "N" (not found), FINDNEXT locates the record whose key logically follows the key passed to FINDOLD.

20.14 FINDPREVIOUS (DEFFN' 212)

FINDPREVIOUS is the same as FINDNEXT except that it finds the previous record in the file, in key sequence.

FINDPREVIOUS may be executed in any situation where FINDNEXT is allowed. This includes following a FINDOLD where Q$ = "N" is returned (key not found), in which case a subsequent FINDPREVIOUS will find the next lowest key.

The calling sequence is:

```
GOSUB'212 (I,P)
```

"I" is the KFAM I.D. Number assigned in an OPEN subroutine.

"P" is the record protect option. P equal to 1 or 3 indicates record protection; P equal to 0 or 2 indicates no record protection.

FINDPREVIOUS Return Codes

Q$ = "X" file not opened or a current key is not defined because of no previous access or a prior error condition.
Q$ = "E" end of file. In this case the dummy key marking the beginning of the file has been reached.
Q$ = "B" busy signal. The User File sector found has been protected by another station. In the case of a busy signal, FINDPREVIOUS goes back to the original key, so that a subsequent FINDPREVIOUS will find the same record again that caused the busy signal. A $BREAK statement and a retry are recommended.
Q$ = "blank" indicates successful completion.

20.15 RELEASE (DEFFN' 238)

The RELEASE subroutine turns off the record protect flag previously set by the calling station.

Any call to a KFAM-7 subroutine for a particular file turns off any protect flag for that file. RELEASE should be used only if there may be a long delay before the next KFAM-7 subroutine is called.

The calling sequence is:

```
GOSUB'238 (I)
```

"I" is the KFAM I.D. Number assigned to the file in an OPEN subroutine.

RELEASE Return Codes

Q$ = "X" if file not open.
Q$ = " " (space) after successful execution.

20.16 RE-OPEN (DEFFN' 213)

RE-OPEN changes the access mode of a currently open file.

The calling sequence is:

```
GOSUB'213 (I,A,P$)
```

"I" is the KFAM I.D. Number.
"A" is the new access mode (1, 2, 3, or 4).
"P$" is the file password.

RE-OPEN Return Codes

Q$ = "A" if the file could not be opened in the new access mode because of an access mode conflict. It remains open in the previous access mode.
Q$ = "P" if the password (P$) was invalid.
Q$ = "X" if file not open or if the access mode is invalid.

20.17 WRITE RECOVERY INFORMATION (DEFFN' 214)

WRITE RECOVERY INFORMATION writes recovery information (most of the KDR) to the next-to-last sector of the User File. Whenever FINDNEW is used, RECOVERY INFO should be rewritten either automatically when the CLOSE subroutine is called or via WRITE RECOVERY INFO before closing a KFAM file. This operation is normally done when the file is closed if using the KFAM0107 module or if the CLOSE WITH RECOVERY INFORMATION option was chosen during BUILD SUBROUTINE MODULE. The file remains open upon completion.

The calling sequence is:

```
GOSUB'214 (I)
```

"I" is the KFAM I.D. Number.

WRITE RECOVERY INFORMATION Return Codes

Q$ = "X" if file not open.
Q$ = "blank" if executed successfully.

20.18 CLOSE (DEFFN' 239)

The CLOSE subroutine is used to close a currently open User File and its companion Key File. The KFAM I.D. Number assigned to a closed file can then be reassigned to another file in an OPEN routine. Also, the file numbers assigned to a User File and Key File can be reassigned. CLOSE alters the access table information to indicate that the station no longer has the file open. The CLOSE subroutine also saves certain critical information when operating in the Shared or Exclusive access modes for the KEY FILE RECOVERY utility, unless CLOSE WITH RECOVERY INFORMATION was not included when (and if) BUILD SUBROUTINE MODULE was used. The calling sequence is:

```
GOSUB' 239 (I)
```

"I" is the KFAM I.D. Number assigned to the file in an OPEN routine. Following execution of the CLOSE routine, this number can no longer be used to access the User File and its associated Key File.

CLOSE Return Codes

Q$ = "X" if file not open.
Otherwise, Q$ = " " (space).

> NOTE:
>
> CLOSE must be executed promptly at the conclusion of file access operations.

20.19 NON-KFAM FILE SUBROUTINES

Please note that the following subroutines are described in Chapter 28 of this manual. These subroutines are available for non-KFAM data files and provide similar access mode capabilities. These subroutines include the following:

```
DEFFN'217  OPEN a new or existing file.
DEFFN'218  Write an END record in a file.
DEFFN'219  CLOSE an open file.
```

CHAPTER 21

THE KFAM REORGANIZE UTILITIES

21.1 INTRODUCTION

There are two KFAM reorganize utilities: REORGANIZE SUBSYSTEM and REORGANIZE IN PLACE. The functions performed by each are nearly identical, that is, all records are reordered within the output file according to their ascending key values, and space previously occupied by deleted records is available following reorganization.

REORGANIZE SUBSYSTEM is a program-controlled standalone routine that is called by a short user-written set-up program. The set-up program provides reorganization parameters (e.g., file names and disk addresses) and optionally loads a program following reorganization. The contents of the input User File and Key File are copied during reorganization to a "reorganized" output User File and Key File. REORGANIZE SUBSYSTEM does not appear on the KFAM utilities menu.

REORGANIZE IN PLACE is an operator-controlled utility program chosen from the KFAM-7 utilities menu. The operator enters the reorganization parameters in reply to prompts. Because the contents of the User File and Key File are copied "in place" within the same User File and Key File, backup copies of both the User File and Key File must be made prior to running this program. Any errors encountered usually destroy the KFAM file.

REORGANIZE SUBSYSTEM is about four to five times faster than REORGANIZE IN PLACE and does not require backup copies. With the exception of large files where there is insufficient disk(ette) space available for copying the output User and Key File, REORGANIZE SUBSYSTEM is generally preferred.

21.2 REORGANIZE SUBSYSTEM STANDALONE ROUTINE

The REORGANIZE SUBSYSTEM is a standalone routine that performs the following KFAM file maintenance operations:

1. Based on an input KFAM file (Key File and User File) it constructs a new output User File which contains active records written into the output User File in ascending order of their key values. Both User Files are opened with Exclusive access.

2. Creates a Key File based on the new output file. Optionally, the new Key File may occupy the same physical space as the input Key File, overwriting the input Key File.

3. Optionally, the new output User File may be copied back to the disk area occupied by the input User File, overwriting the input file.

The modules which comprise REORGANIZE SUBSYSTEM are as follows:

ISS.REFB	ISS Reference File
KFAM3507	Start up, open files
KFAM3607	Generate Code
KFAM3707	Reorganize Parts 1 and 2
KFAM3907	Reorganize Part 3, close files

REORGANIZE SUBSYSTEM modules may be copied to the desired disk(ette) using the supplied ISS reference file ISS.REFB during Copy/Verify (utility) operation. Specify INPUT MODE = INDIRECT to use a reference file during Copy/Verify operation.

Writing The Set-up Module

The user-written set-up program which provides reorganization parameters can be broken down into two parts.

1. Lines 1-3499 contain statements executed before the REORGANIZE SUBSYSTEM is loaded. These lines must clear the CRT screen, select disk file devices, and load KFAM3507. These lines must be cleared by the LOAD DC statement. They can include additional preprocessing, if desired.

2. Lines 4200-4799 contain statements which assign reorganization parameters to specific variables. They remain as an overlay to the first reorganization module. They are executed after the first reorganization module defines its common variables and sets default values.

The master set-up program is shown below. A line may be omitted if the default value shown is the desired value. Read all comments before writing a set-up module.

Line	Contents	Default Value	See Comment
10	REM program identification		
20	PRINT HEX(03)	-	1
50	SELECT DISK (disk address for REORGANIZE SUBSYSTEM disk)	-	
60	SELECT #1 input User File device address	-	
70	SELECT #2 input Key File device address	-	
80	SELECT #3 output User File device address	-	2
90	SELECT #4 output Key File device address	-	3
100	SELECT #5 user program device address	-	4
110	LOAD DC T#0, "KFAM3507" 10,4199	-	5
4210	N1$ = input User File name	-	
4220	P1$ = input User File device address as "xyy"	-	6
4230	N2 = input Key File number	1	7
4240	P2$ = input Key File device address as "xyy"	-	
4250	N3$ = output User File name	-	8
4260	P3$ = output User File device address as "xyy"	-	
4270	03$ = "C" catalog output User File if uncataloged = "Y" output User File already cataloged, do not catalog it. = "N" output User File is not already cataloged, catalog it.	C	9
4280	03 = number of sectors to allocate to output User File. This statement not needed if 03$ = "Y" (above). *If the utility catalogs the file, the default value for 03 is the number of sectors in the input User File.	*	9
4290	06$ = "C" to copy back output User File over input User file when reorganization completed. = blank to leave input User File intact.	blank	10
4300	N4 = output Key File number	1	11
4310	P4$ = output Key File device address as "xyy"		
4320	04$ = "C" catalog output Key File, if uncataloged. = "Y" output Key File cataloged, do not catalog it. = "N" output Key File not cataloged, catalog it.	C	12

4330	O4 = number of sectors to allocate to the output Key File. Omit this if O4$ = "Y". *If the utility catalogs the file, the default value is calculated in proportion to the input Key File size times the increase or decrease in User File size.	*	12
4340	N5$ = name of program to be loaded following reorganization = blank - no program to be loaded	blank	13
4350	P5$ = device address of user program as "xyy"		
4360	P$ = input User File password	blank	14
4370	P9$ = output User File password	blank	14
4380	S2 = Station (partition) number *ISS start-up common variable for this station is the default value.	*	
4390	For type "C" files previously written in BA mode, set M6=1		

Comments

1. The CRT screen should be cleared prior to calling the REORGANIZE SUBSYSTEM. Lines 0-3 are used by the routine for messages. Lines 4-15 may be used for a user-written display. For example, line 20 might be:

```
20 PRINT HEX(030A0A0A0A); "REORGANIZE INVENTORY FILE."
```

2. The output User File device address may be the same as the input User File device address if the two files have different names.

3. If the output Key File device address is the same as the input Key File device address and the output Key File name is the same as the input Key File name, then the output Key File replaces the input Key File. See comment 10.

4. If the REORGANIZE SUBSYSTEM is to call another program when it completes execution, the device address of this program file must be selected for file number #5.

5. The last statement to be executed in the range 1-3499 must be a LOAD DC that loads module 1 of the utility and clears lines 1-3499 as it does so. The module name is "KFAM3507".

6. Example:

```
4220 P1$ = "B10"
```

7. This number is assigned during INITIALIZE KFAM FILE and appears as the 6th character in the input Key File name (normally, it is 1).

8. The output User File name need not conform to the KFAM file naming conventions. This relaxation of normal KFAM requirements may be useful if the "copy back" option is chosen (line 4290) since, in this case, it may be desirable to use an established work file that may have any name.

9. If "C" is assigned to variable 03$, the output User File is cataloged by this utility if the output file does not already exist. "C" is the default value of variable 03$ (line 4270).

   If "N" is assigned to variable 03$, the system ensures that the named output User File does not already exist on the disk and catalogs the output User File.

   If the utility catalogs the output User File, it allocates to it the same number of sectors that are in the input User File unless a different number is specified by assigning the desired number of sectors to variable 03.

   If "Y" is assigned to variable 03$, the system checks that the named output file already exists. Variable 03 need not be assigned a value.

   The output User File must contain at least 10 sectors.

10. If variable 06$ is assigned the value "C" (line 4290), then the utility constructs the output Key File name from the input Key File name and copies the output User File back into the input User File area, overwriting the input User File. If variable 06$ is assigned a blank, then the output Key File name is constructed from the output User File name and the output file is not copied back.

11. This number is used in the construction of the output Key File name, in which it becomes the 6th character. If the constructed name is the same as the input Key File name and the same address is specified for both input and output Key Files, then the output Key File replaces the input Key File.

12. The effect of these responses for variables 04$ and 04 is analogous to variables 03$ and 03 discussed in comment 9. However, if the utility catalogs the Key File, its size is proportional to the input Key File size times the increase or decrease in User File size.

13. If N5$ is assigned a program name, the program is loaded upon completion of the utility. The program must reside at the address SELECTed for file number #5 (line 100).

14. The input User File Password, and the output User File Password if the output User File is already cataloged, is checked against variables P$ and P9$ respectively. These values must be identical or an error message appears. If the output User File is not cataloged and if P9$ is blank, then a Password of blanks is assigned to the User File. With an uncataloged output User File, the Password assigned is later required when the file is opened. For example, if P9$ contains anything other than blanks, then that Password assigned must be provided for any station to Open that file.

Shown below is an example set-up program.

```
10 REM EXAMPLE OF A REORGANIZATION SET-UP PROGRAM FOR KFAM-7
20      PRINT HEX(030A0A0A0A); "REORGANIZE INVENTORY FILE"
50      SELECT DISK 310 :REM REORG. SUBSYSTEM
60      SELECT #1/320   :REM INPUT USER FILE
70      SELECT #2/B20   :REM INPUT KEY FILE
80      SELECT #3/320   :REM OUTPUT USER FILE
90      SELECT #4/B20   :REM OUTPUT KEY FILE
100     SELECT #5/310   :REM USER PROGRAM
110     LOAD DC T#0, "KFAM3507" 10,4199
4210    N1$ = "INVTF040"
4220    P1$ = "320"
4240    P2$ = "B20"
4250    N3$ = "WORK"
4260    P3$ = "320"
4290    06$ = "C" :REM COPY BACK OUTPUT USER FILE
4310    P4$ = "B20"
4340    N5$ = "START"
4350    P5$ = "310"
```

REORGANIZE SUBSYSTEM Operation

The operation of REORGANIZE SUBSYSTEM may be divided into three parts:

1) The User File is read sequentially, using FINDFIRST/FINDNEXT, and copied to the output file so that the records are physically in sequential order, and DELETED records are eliminated.

2) A new Key File is built, based on the keys in the output User File, using a special procedure. The new Key File, optionally, may occupy the same physical space as the old Key File.

3) If indicated by the set-up program, the output User File is copied back to the input User File, overwriting the original.

The original Key File and User File are not altered until the output User File has been written, complete with the necessary information to restore the Key File. Therefore, it is not essential to have backup copies of the User File and Key File. If the system fails during Part 1 of the reorganization, the original Key File and User File are intact. If the system fails during Part 2, both the input User File and the output User File are intact, and a Key File may be built for either one, using the KEY FILE RECOVERY utility. During Part 3, the output User File remains intact, as well as the Key File. Although backup disks are not necessary for this operation, it is good practice to make backup copies regularly, especially of the User File.

There are no operating instructions for this program because normally no operator intervention is required. However, there are recovery procedures for certain error conditions. These are described in Appendix A.

21.3 REORGANIZE IN PLACE UTILITY PROGRAM

REORGANIZE IN PLACE reorganizes a KFAM File in place. The record which belongs first, in ascending key sequence, is switched with the record that is physically first. This process is repeated for the second record, and so forth, until the entire User File has been placed in sequential order. In the process, all DELETED records are removed. Then, the Key File is reinitialized and a new Key File is created in the space formerly occupied by the old Key File. No additional disk space is required for the User File or the Key File; a 15-sector work file, KFAMWORK, is required. This program is 4 to 5 times slower than the REORGANIZE SUBSYSTEM utility and, therefore, should be used only if the file is too large to permit simultaneous mounting of an output file as required by the REORGANIZE SUBSYSTEM.

Because this program destroys both the User File and the Key File as they formerly existed, there is no possible recovery in the event of hardware or software error. For that reason, backup copies of the User File and the Key File must be made before running this program.

This program reorganizes any KFAM file, with the exception of type M records with more than 40 sectors per record. However, it should be noted that multiple-sector records require extra storage space in memory, and that this program cannot operate in a 9K system if the record length exceeds 8 sectors.

Operating Instructions

	DISPLAY	INSTRUCTIONS
1.		1. Make backup copies of both the User File and the Key File. If anything goes wrong during the execution of the utility, both files are destroyed.
2.		2. Mount the disk platter(s) containing the User File and the Key File.
3.		3. To load the REORGANIZE IN PLACE utility, touch the specified Special Function key in reply to the KFAM-7 menu.
4.	ARE THERE BACKUP COPIES OF USER FILE AND KEY FILE? (Y OR N)	4. Enter Y if backup copies exist. Proceed with Step 6. below. Enter N for "no" or "don't know". Proceed with Step 5.
5.	The system displays: ANY ERROR DURING THE RUNNING OF KFAM3207 WILL DESTROY BOTH FILES. MAKE COPIES OF THE DISK PLATTER(S) CONTAINING THE USER FILE AND THE KEY FILE BEFORE RUNNING THIS PROGRAM. STOP	5. Make backup copies of both the User File and the Key File. Key CONTINUE and RETURN. Go to step 4.
6.	ENTER USER FILE NAME (SSSSFJNN)	6. Enter the name of the User File.

NOTE:

Error messages and recovery procedures are located in Appendix A.

7.	ENTER USER FILE DEVICE ADDRESS	7. Enter the disk device address of the User File (xyy form).
8.	ENTER PASSWORD	8. Enter the Password assigned to the User File. If no Password assigned, key RETURN without entering any characters.
9.	ENTER KEY FILE NUMBER (NORMAL=1)	9. Enter the Key File Number. The Key File Number should always be 1, unless there are multiple key files for a single User File, in which case, the Key File Number can be any digit from 1 to 9.
10.	ENTER KEY FILE DEVICE ADDRESS	10. Enter the disk device address of the Key File (xyy form).
11.		11. The system opens the Key File and User File and begins processing. The file is reorganized. No operator intervention is required from this point on.
12.		12. The KFAM menu appears upon completion.

CHAPTER 22

REALLOCATE KFAM FILE SPACE UTILITY PROGRAM

22.1 OVERVIEW

REALLOCATE KFAM FILE SPACE and the ISS utility COPY/VERIFY can be used in conjunction with one another to lengthen or shorten KFAM Key Files and User Files. After a KFAM file or a number of KFAM files have been copied using COPY/VERIFY, REALLOCATE KFAM FILE SPACE must be run. See Section 18.5 for KFAM file copy procedures.

An alternative to copying a file with COPY/VERIFY and then using REALLOCATE KFAM FILE SPACE is to run REORGANIZE SUBSYSTEM, which allows its user to reorganize, copy, change file space allocation, and change the file name--all at the same time.

22.2 OPERATING INSTRUCTIONS

	DISPLAY		INSTRUCTIONS
1.		1.	Mount the disks containing the User File and Key File. From the KFAM menu, load REALLOCATE KFAM FILE SPACE by touching the indicated Special Function Key.
2.	ENTER USER FILE NAME (SSSSFJNN)	2.	Enter the name of the file.

NOTE:

Error messages and recovery procedures appear in Appendix A. Although not shown here, prompting dashes accompany the actual prompts.

3.	ENTER USER FILE DEVICE ADDRESS	3. Enter the xyy form of the User File device address.
4.	ENTER PASSWORD	4. An entry to this prompt is only required if the User File has a Password assigned to it, in which case the Password must be entered exactly as previously assigned to this User File. If a Password was not assigned, key RETURN without entering any characters.
5.	ENTER KEY FILE NUMBER (NORMAL=1)	5. Normally, enter 1. If there is more than one Key File associated with this User File, enter the number of the Key File to be accessed.
6.	ENTER KEY FILE DEVICE ADDRESS	6. Enter the xyy form of the Key File disk device address.
7.		7. The system reads the KDR and associated entries and then makes the internal adjustments necessary for the User File and Key File.
8.		8. The KFAM menu appears upon completion.

CHAPTER 23

PRINT KEY FILE UTILITY PROGRAM

23.1 OVERVIEW

The PRINT KEY FILE utility program prints the contents of the selected KFAM Key File to the ISS printer address. The User File access table for each station (1-16) is also printed. The Key File contains one Key Descriptor Record (KDR) and the Key Index Records (KIR's) for this KFAM file. The User File, which contains the access table, is accessed in the Read-Only access mode, thus the User File disk address and its Password (if any) must be entered. The Key File also must be on-line and is accessed in the Read-Only mode.

Information printed indicates the access mode number (ACCESS TYPE=1, 2, 3, or 4) and related Key File information. The station running PRINT KEY FILE, which operates in the Read-Only access mode, should have an ACCESS TYPE=2 next to its station number. This information is extremely useful in determining the cause of long delays in accessing files. PRINT KEY FILE may be followed by RESET ACCESS TABLE if necessary or the RELEASE and/or CLOSE subroutines executed from the appropriate station number(s) to close a KFAM file accidentally left open.

Completion codes and protected sectors are always indicated as "Z" and "FFFF" respectively, irrespective of the corresponding values in global memory.

23.2 OPERATING INSTRUCTIONS

DISPLAY	INSTRUCTIONS
1.	1. From the KFAM-7 menu, load the PRINT KEY FILE utility by touching the indicated Special Function Key.
2. ENTER USER FILE NAME (SSSSFJNN)	2. Enter the name of the User File with which the Key File is associated.

			NOTE: Error messages and recovery procedures appear in Appendix A. Also, although not shown here, dashes accompany most prompts for easy parameter entry.
3.	ENTER USER FILE DEVICE ADDRESS	3.	Enter the xyy form of the User File disk device address.
4.	ENTER PASSWORD	4.	An entry to this prompt is only required if the User File has a Password assigned to it, in which case the Password must be entered exactly as previously assigned to this User File. If a Password was not assigned, key RETURN without entering any characters.
5.	ENTER KEY FILE NUMBER (NORMAL=1)	5.	Normally, enter 1. If there is more than one Key File associated with this User File, enter the number of the Key File to be accessed.
6.	ENTER KEY FILE DEVICE ADDRESS	6.	Enter the xyy form of the Key File disk device address.
7.		7.	The access table and then the Key File's contents are printed.
8.		8.	The KFAM-7 menu appears.

CHAPTER 24

KEY FILE RECOVERY AND RESET ACCESS TABLE UTILITY PROGRAMS

24.1 KEY FILE RECOVERY

If a Key File is destroyed, the Key File Recovery utility permits it to be fully reconstructed from the data in the User File, provided application programs that operate on the file adhere to the following conventions:

1) All DELETED records and any blocked records in the last active sector that are inactive must be flagged in the User File with HEX(FF) in the first byte of the key.

2) With KFAM-7, CLOSE is a system requirement. If BUILD SUBROUTINE MODULE was used and the CLOSE WITH RECOVERY INFO option was not selected, the user must include the WRITE RECOVERY INFO subroutine and must execute the WRITE RECOVERY INFO subroutine before closing a KFAM file whenever FINDNEW has been used.

If the Key File already exists on the designated disk, this utility reuses that file; otherwise, it catalogs a new file with sufficient space to index the maximum number of records in the User File. The User File is opened in the Exclusive mode and the recovery information (most of the KDR) in the next-to-last sector is accessed.

If more than one Key File exists for this one User File, it may be impossible to use this utility.

The utility uses the ISS start-up printer address to list duplicate keys.

The output report of the Key File indicates the access mode (ACCESS TYPE) for each station accessing the file and any records protected by any station.

Operating Instructions: KEY FILE RECOVERY

	DISPLAY		INSTRUCTIONS
1.		1.	Mount the disk containing the User File and the disk to contain the reconstructed Key File.
2.		2.	From the KFAM-7 menu, load KEY FILE RECOVERY by touching the indicated Special Function Key.
3.	ENTER USER FILE NAME (SSSSFJNN)	3.	Enter the name of the User File for which the Key File is to be reconstructed.
			NOTE: Error messages and recovery procedures follow in Appendix A.
4.	ENTER USER FILE DEVICE ADDRESS	4.	Enter the xyy form of the User File disk device address.
5.	ENTER PASSWORD	5.	An entry to this prompt is only required if the User File has a Password assigned to it, in which case the Password must be entered exactly as previously assigned to this User File. If a Password was not assigned (Password = blanks), key RETURN without entering any characters.
6.	ENTER KEY FILE NUMBER (NORMAL=1)	6.	The Key File Number should always be 1, unless there are multiple Key Files for a single User File, in which case the Key File Number may be any digit from 1-9.
7.	ENTER KEY FILE DEVICE ADDRESS	7.	Enter the xyy form of the Key File disk device address.

8. IS KEY FILE CATALOGED (Y OR N)	8. If the Key File is cataloged at the address given in the previous step, enter Y; otherwise, enter N. Execution error messages are printed or displayed on lines 7-10 respectively, if the ISS printer address indicates a printer (e.g., 215) or is 000. With displayed output, if an error message is encountered, key CONTINUE and RETURN to continue.
9.	9. The system, which has now recreated the Key File, returns the KFAM menu to the screen.

24.2 RESET ACCESS TABLE

For KFAM-7 there is an access table located in the User File and access tables in the KDR and in global memory consisting of Key File control information. For each possible station, an entry is maintained in the User File's access table for that station's current access mode for this file. If a station is accessing the file, either 1,2,3, or 4 appears, depending on the access mode; otherwise, a blank appears if the station is not currently accessing the file. The User File and KDR access tables may be printed by the PRINT KEY FILE utility.

For each station (1-16) the ACCESS TYPE listed by PRINT KEY FILE shows the access mode for each station accessing the file. The numbers are as follows:

1 - Inquiry	does not allow Exclusive file access.
2 - Read Only	does not allow Shared or Exclusive file access.
3 - Shared	does not allow Read Only or Exclusive file access.
4 - Exclusive	does not allow any other file access.
NONE -	This file is not currently open by any station.

With this information provided by PRINT KEY FILE, one can determine which station has the file open in the access mode that conflicts with the access mode desired. If that station is not accessing the file, the entry in the access table may be caused by the file not being closed by that station, thus creating the "phantom" entry in the access table, which must be corrected by running RESET ACCESS TABLE.

This utility is provided to reset all of the access table information to blanks in the event of a program failure or system failure that has left the access table with erroneous non-blank characters. The utility also turns off any "record protect flags" that may be left on.

CAUTION:

This utility should not be run if any other station is currently accessing the file. The utility has no way of knowing whether entries in the access table are "live" or "dead", and resets all access table bytes to blanks indiscriminately. Before running this program, the user absolutely must check to make sure that no other station is currently accessing the file. Otherwise, there could be an unpredictable scrambling of results.

Operating Instructions: RESET ACCESS TABLE

	DISPLAY		INSTRUCTION
1.		1.	From the KFAM-7 menu, load RESET ACCESS TABLE by touching the indicated Special Function Key.
2.	ENTER USER FILE NAME (SSSSFJNN)	2.	Enter the User File Name. Error messages appear in Appendix A.
3.	ENTER USER FILE DEVICE ADDRESS	3.	Enter the xyy form of the User File disk device address.
4.	ENTER PASSWORD	4.	An entry to this prompt is required only if the User File has a Password assigned to it, in which case the Password must be entered exactly as previously assigned. If a Password was not assigned, key RETURN without entering any characters.
5.	ENTER KEY FILE NUMBER (NORMAL=1)	5.	Normally, enter 1. However, if there exists multiple Key Files for this single User File, enter a single digit from 2-9.
6.	ENTER KEY FILE DEVICE ADDRESS	6.	Enter the xyy form of the Key File disk device address.
7.		7.	The system resets this KFAM file's access table.
8.		8.	The KFAM menu appears.

CHAPTER 25

THE KFAM CONVERSION UTILITY PROGRAMS

25.1 OVERVIEW

KFAM-7 includes two KFAM conversion utilities. These are CONVERT KFAM-3 to KFAM-7 and CONVERT KFAM-4 to KFAM-7. They are provided to convert a file created under one of the earlier KFAM's to the format of KFAM-7. The procedures and operating instructions for these programs are the same regardless of which one is being used.

The structure and format of the Key File is not the same for KFAM-7 as for KFAM-3 or KFAM-4. Therefore, the conversion process consists of dropping the old Key File and creating a new Key File in the KFAM-7 format. The Exclusive access mode is used.

To convert a file to KFAM-7, the procedure is as follows:

1. Backup copies should be made of the User File and the old Key File.

2. Mount the User File and the old Key File.

3. Execute the appropriate "CONVERT" program.

The User File and Key File are now converted and can be accessed via KFAM-7.

25.2 OPERATING INSTRUCTIONS

1.	1. From the KFAM-7 menu, load the appropriate CONVERSION utility by touching the specified Special Function Key.
2. ENTER USER FILE NAME (SSSSFJNN)	2. Enter the name of the User File to be converted.

			NOTE: Error messages and recovery procedures appear in Appendix A. Also, although not shown here, dashes accompany most prompts for easy parameter entry.
3.	ENTER USER FILE DEVICE ADDRESS	3.	Enter the xyy form of the User File disk device address.
4.	ENTER PASSWORD	4.	To require that any station wishing to access this file must first provide a Password, enter the Password to be assigned to the KFAM-7 file being created. If a Password need not be assigned to this file, key RETURN without entering any characters (Password = blanks).
5.	ENTER KEY FILE NUMBER (NORMAL=1)	5.	Normally, enter 1. However, if there are multiple Key Files for a single User File, enter a digit from 2-9.
6.	ENTER KEY FILE DEVICE ADDRESS	6.	Enter the xyy form of the Key File disk device address.
7.		7.	The utility converts the specified KFAM file to KFAM-7 format.
8.		8.	The KFAM menu appears.

CHAPTER 26

GENERAL TECHNICAL INFORMATION

26.1 KEY FILE RECORD LAYOUT AND STORAGE IN MEMORY

The first sector of the Key File contains the Key Descriptor Record (KDR). The KDR is rewritten to the Key File when a FINDNEW, FINDNEW(HERE), DELETE, or CLOSE subroutine is executed. The remaining sectors contain Key Index Records (KIR's), as many as are necessary to index the User File up to eight index levels. An END record follows the last KIR sector. A hardware catalog trailer occupies the last sector for the file.

Key Descriptor Record (KDR)

KDR contents occupy the first sector of the Key File, as well as the next-to-last sector of the User File (bytes 3-146, T$2()) except for variable V, which is always zero (0) in the KDR on disk.

The value of variable V stored in memory indicates the subroutine being executed, except for its use as a working variable during OPEN, as follows:

```
1 =   busy, FINDNEXT
2 =   busy, FINDPREVIOUS
4 =   default, FINDOLD, FINDFIRST, FINDLAST, RELEASE
5 =   RE-OPEN, WRITE RECOVERY INFO, CLOSE
7 =   busy, FINDNEW(HERE)
8 =   busy, FINDNEW
```

T$(3) 48 contains KDR information and is packed as follows:

START	LENGTH	$UNPACK VARIABLE	IMAGE	CODE	
1	16	-	-	-	Indicates completion codes per station (1-16) except: Initialized to "2" by some utilities (KFAM1007, KFAM7007). Following FINDOLD, not found, = "2". Following FINDNEW, FINDNEW(HERE), or DELETE, all values of "9" or less are set to "3". Following a successful Open, set to letter "0". Following successful CLOSE, set to "2". If Q$=B, set to "9".
17	32	-	-	-	Protected sectors per station. HEX(FFFF) if no protected sector.
49	1	T0	5001	X,P	Number of index levels.
50	2	T2$2	A002	X,P	Relative sector address, highest level index sector.
52	2	Q2$2	A002	N,P	User File sectors used, minus 1.
54	2	V2$2	A002	N,P	Key File sectors used, minus 1.
56	4	T8	5004	N,P	Count of active records in the file.
60	1	V6$1	A001	N	Sectors per logical record.
61	2	V3$2	A002	N	Key File, last available relative sector.
63	2	Q3$2	A002	N	User File, starting relative sector of last available record position.
65	1	V8$1	A001	N	Records per block. No comma follows this.
66	6	-	-	-	Per byte: 1 = record type is A,B,C,M, or N. 2 = record length. 3,4 = starting position of key. 5 = key length. 6 = number of entries in KIR.

97	*	T3$3	-	S	User File, current relative sector for FINDNEW, per station, in first two bytes; also, current record within block for FINDNEW, per workstation, in last byte (was Q4$ and V5$).

All numbers are hex except T0 and T8 in the preceding table of the KDR.

CODES in the preceding table are indicated as follows: X=unpack always, N=unpack for FINDNEW and DELETE, P=pack for FINDNEW and DELETE, S=unpack and pack, FINDNEW, for this station only.

* T3$ is stored per station. The location is STR(T$(3), 3*S2-2, 3) where S2=station number, in the preceding table.

Key Index Record (KIR)

Variable Name	Bytes On Disk	Contents
T9$2	3	Sector address (hex), this sector, relative to first sector of Key File = 0.
T0$(4)60	244	A 240-byte array containing KIE's. Number of KIE's per KIR can vary from 7 to 48. Unused KIE's are filled with all bytes HEX(FF). Active KIE's are packed as follows: K bytes: key 3 bytes: pointer Pointer points to next lower index level KIR or User File record if lowest level. The first two bytes of the pointer contain the sector address (hex) relative to the start of the file. The last byte contains the record number (hex) within the sector if the pointer is to a data record, and is not defined if the pointer is to a lower level KIR.
TOTAL	247 bytes	

Internal Storage: User Partition

Certain information also stored in the KDR is stored internally in user partition memory for the last file accessed.

Variables for KFAM files which are open but not the last file accessed are stored in two arrays, T5$() and V0$(). The array element is the KFAM I.D. Number. Variables are packed in field format.

V0$(3)21 = variables set at OPEN, remain unchanged. (NOTE: array element may be set to the number of files to be opened.)
@Q8$20 = hex image for unpacking.

Variable	Start	Image	Description
V0$2	1	A002	Absolute starting sector of Key File.
V6	3	5002	Internal File I.D.
V4$4	5	A004	Hex image for unpacking entry from KIR, HEX(A0XXA003), where XX = key length.
V0	9	5001	Access Mode (1,2,3, or 4).
T1	10	5002	File #, Key File.
T2	12	5002	File #, User File.
T4	14	5002	Key length.
T5	16	5002	KIE length = T4 + 3.
V7	18	5002	KIR bytes used = INT (240/T5)*T5.
V1	20	5002	Last key location = V7 - T5 + 1.

T5$(3)58 = variables which change with each access, saved every time files are switched (NOTE: array element may be set to the number of files to be opened. T5$(3)58 indicates 3 KFAM files maximum open to this station.)
@T5$10 = hex image for unpacking.

Variable	Start	Image	Description
T4$3	1	A003	Pointer to record accessed, bytes: 1 - 2 = relative sector address within user file. 3 = record within block.
T7$30	4	A01E	Last key accessed.
T8$1	34	A001	Internal completion code.
T$8	35	A008	Path to last record accessed, pointers to entry within KIR.
T2$(8)2	43	A002	Path to last record accessed, KIR relative.

Internal Storage: Global Memory

KFAM-7 controls access to the disk internally through tables in the global partition rather than by reading and writing the KDR, as is done in KFAM-5. This saves disk access time and also saves time by not hogging the disk except on OPEN, RE-OPEN, CLOSE, and WRITE RECOVERY INFO. All KFAM accesses go through the same global partition "KFAM". It is important that the KFAM-7 subroutines should not reside in more than one global partition because this module (KFAM0107) not only provides subroutines but also coordinates record access to KFAM files.

In the global area is a table of open files, @T$(30)14. The number of table entries can be increased or decreased from 30, depending on the maximum number of KFAM files that can be open at any one time, by all stations on a given system. (Global memory must be increased if more than 30 KFAM files are to be open.) The contents of this table, per entry, are as follows:

START	LENGTH	CONTENTS
1	2	KDR sector (V0$) = starting sector of Key File.
3	1	Device address of Key File, compressed. Bytes 2 and 3 of device address are packed, then OR'ed with 80 if byte 1 = "B".
4	1	Number of index levels, T0, in IBM packed format.
5	2	Relative sector address of highest level index sector, T2$.
7	8	Per station 1 - 16, one half byte for internal completion code.

Note that the first 3 bytes above form a unique identifier for the particular Key File being accessed.

Also note that the internal completion code is used for two purposes: (1) to determine whether a particular file is open or closed to a particular station and (2) to determine whether internal variables T0 (number of index levels), T2$ (sector address of highest level index), T2$() (path through index to current record, in terms of KIR sectors read), and T$ (path to current record, in terms of KIE starting location within KIR) are all currently valid.

Bit settings within the current completion code are as follows:

0	normal completion, above variables valid.
1	path not defined (T2$() and T$), KDR OK.
2	path not defined (T2$() and T$), reread KDR (changed by another station).
4	index level added, get new values of T0, T2$ from table @T$(), above.
8	error condition, next and previous records not defined.
E	file open.
F	file closed.

The bit settings above are unpacked into T8$. In access modes 1 (Inquiry) and 3 (Shared), internal completion codes are packed into table @T$(). In access modes 2 (Read Only) and 4 (Exclusive), the only values appearing in table @T$() are "E" for open or "F" for closed.

An internal file ID, V6, is maintained by KFAM-7 to indicate the number of the entry for a given file within @T$(). @T$(V6) is the table entry for the particular file. The internal file ID should not be confused with the KFAM ID, the Key File Number, or the file numbers of the Key File and User File in the Device Table (the latter are specified by the user and OPEN parameters). The internal file ID is another number which is assigned by KFAM-7 for its own internal use when the file is opened.

In the non-interactive modes (2=Read Only, 4=Exclusive), KFAM-7 simply makes an entry in table @T$() to indicate that the file has been opened. It also stores T0 and T2$ in the table @T$ (V6) to save reading the KDR when files are switched. It then operates very much as KFAM-3, where no interaction is possible, because in Exclusive mode no other station can access the file, and in Read Only mode no other station can change the Key File.

The interactive modes are defined as 1 = Inquiry and 3 = Shared. KFAM-7 maintains a queue to regulate access to files in the interactive modes. The queue contains two entries, the station number (S2), in hex in one byte of @Q$, and the internal file ID (V6), in hex in the corresponding byte of @Q9$. Upon entry to a KFAM-7 subroutine in an interactive mode, the station number and the internal file ID are placed at the end of the queue. The queue (@Q9$) is then searched for the internal file ID, and the station number in the corresponding position of @Q$ has access to the file. All other stations requesting the file must wait. When the station accessing the file is finished, its entry is dropped from the queue, and the next station in the queue requesting that file is allowed to access it.

The queue allows access to the file on a first-in-first-out (FIFO) basis, i.e., first-come-first-served. Under KFAM-7, access is allowed from the various stations in the order requested. The KFAM-7 queue is not to be confused with the delegation of program execution time between different partitions under the control of the 2200MVP itself. The queue is provided by KFAM-7 and is used only for KFAM-7 files.

There is only one queue for all stations requesting all files. This does not inhibit different stations from accessing different files at the same time. For example, if the queue looks like this:

```
@Q$(station) 1  3  5  4  2
@Q9$(file)   4  2  1  2  3
```

Station 1 can access file 4, station 3 can access file 2, station 5 can access file 1, and station 4 must wait to access file 2 until station 3 is finished.

KFAM-7 maintains a table of protected sectors, protecting a sector of the User File from access by another station if the protect flag is set (argument symbolic variable, P = 1 or 3). The table of protected sectors is stored in a global variable, @V4$(30)4. This global table contains all protected sectors for all stations accessing all files. The array dimension can be changed upwards or downwards from 30 if necessary. If the table is full, the station requesting sector protection simply waits until there is a vacant slot in the table. Sector protection is only effective in the interactive modes (1 and 3).

The contents of @V4$(), per entry, are as follows:

START	LENGTH	CONTENTS
1	1	Station number, S2, hex
2	1	Internal file ID, V6, hex
3	2	Protected sector - STR(T4$,,2)

> NOTE:
>
> If a User File is being accessed by two or more Key Files, a protected sector as accessed through one Key File will not be recognized by a subroutine accessing the file through another Key File. KFAM does not support multiple Key Files per User File, but it is possible to design a protection scheme external to KFAM.

Through the use of these internal tables, reading and writing the KDR and hogging the disk are kept at a minimum. Instead of the KDR being the communications link between different stations accessing the same file, this communication information is held internally, thus eliminating disk access time and making throughput more efficient.

Global variables are also used as program constants and working variables, in order to cut down on the space required for KFAM variables in the user partitions. In order to update global variables, the program is hogged at certain critical times. This means that there are certain points in global memory which can only be executed by the one station which has gained access, as opposed to the normal case where the code may be executed at the same time (logically) by several different stations. At these critical points, all stations other than the one which has gained access must wait.

Points at which program execution is hogged are as follows:

Non-KFAM OPEN (217)
Non-KFAM END (218)
Non-KFAM CLOSE (219)
OPEN (230): Setting up table @T$(), executing non-KFAM OPEN (217)

CLOSE (239): Setting table @T$(), executing non-KFAM END (218) and non-KFAM CLOSE (219)
RE-OPEN (213): Executing non-KFAM OPEN (217)
WRITE RECOVERY INFO (214): Executing non-KFAM END (218)
FINDNEW (233) and FINDNEW (HERE) (234):
Whenever a KIR sector is split, or about one time in eight depending on key length and other factors.
Any subroutine: Adding or deleting a queue entry, updating internal completion codes.

The times when the program is hogged are either infrequent or brief, and therefore should not slow down performance very much.

26.2 KEY FILE STRUCTURE

The structure of the Key File is similar to the structure called a B-tree, which is discussed on pages 473-479 of THE ART OF COMPUTER PROGRAMMING: Volume 3/Sorting and Searching, by Donald E. Knuth.

A Key File must permit rapid access to any particular User File record and may also be updated at any time without a major reorganization of the file. The B-tree structure, as modified, satisfies this double requirement.

The structure of the Key File is best described by showing how the file is constructed. The first step in INITIALIZE KFAM FILE is to create one KIR record, which contains one dummy KIE with a key value of binary zero (all bytes HEX(00)). This dummy KIE serves to "prime" the system so that the same program logic can be applied to a null or empty file as is applied to a file containing active records. Being the lowest possible key, it also serves to mark the lower limit of the Key File. For example, FINDFIRST is done by searching for the binary zero key and then doing FINDNEXT. This dummy key can be thought of as the 0th entry in the Key File and represents nothing, except as the marker of the lower boundary.

In the examples below, this dummy key is designated as "000". Please note that the actual value is binary zero, HEX(000000), and not the characters "000" or HEX(303030). The characters "000" may be used as an active key and will not conflict with the dummy key.

The unused KIE's in any KIR always have all bytes set to HEX(FF). Thus the original KIR record has the first key set to all HEX(00) and the remaining keys set to all HEX(FF).

In the examples below, these unused keys are designated as "FFF". Please note that the actual value is HEX(FFFFFF...) and not the characters "FFF" or HEX(464646).

Two items in the KDR record are essential to searching the Key File. One is the number of index levels, T0. To start with, T0 = 1, because there is only one level of index. The other item is the relative sector address of the highest level index, T2$. At the starting point, there is only the one index sector, the KIR record described above, and its sector address is always HEX(0001). (The KDR record always occupies sector HEX(0000) or the first sector of the Key File, and the initial KIR follows it in the second sector at relative address HEX(0001).)

The Key File is now set up to begin entering active KIE's. As new keys are added to the file, the respective KIE's are inserted in the KIR in their proper sequential order. Higher keys are moved up one position, and one HEX(FF) key is dropped off the end.

For example, if the first three keys to be inserted are 276, 913, and 198, the KIE's would be arranged as follows:

```
Start:       000, FFF, FFF, etc.
First Key:   000, 276, FFF, etc.
Second Key:  000, 276, 913, FFF, etc.
Third Key:   000, 198, 276, 913, FFF, etc.
```

Keys are inserted in the first KIR in this manner until it is filled. The number of keys per KIR depends upon the size of the key. Let us assume for this example that the first KIR has been completely filled by one dummy key plus 14 active keys:

```
000, 009, 147, 198, 276, 292, 589, 591, 671, 710, 730,
809, 851, 903, 913
```

At this point the key 796 is to be added. Since there is no room in the one KIR to add another key, the KIR is split in two. A new KIR is created, and the KIE's are divided between the old KIR and the new KIR:

```
Old KIR:  000, 009, 147, 198, 276, 292, 589, 591, FFF, etc.
New KIR:  671, 710, 730, 796, 809, 851, 903, 913, FFF, etc.
```

The new KIR occupies relative sector HEX(0002). Note that the key added, 796, is inserted in its proper sequential order and falls in the new KIR.

With more than one KIR now in the file, the concept of "level" enters in. Both KIR's so far created are on level 1, the lowest level. The lowest level is defined as the level which contains the pointers to the data records in the User File. Whenever a KIR is split, the new KIR is on the same level as the old KIR.

Rather than search the KIR's sequentially for a given key, the system searches via a tree structure. There is one and only one KIR at the highest level. Its sector address is recorded in the KDR. The search is started by reading this sector. Up to this point, the search has been completed by locating the position of the key within the one sector. But at this point, there are two KIR's on level 1, and a higher level index must be created to reference them.

Therefore a third KIR is created. It is a level-2 index and contains two keys, 000 and 671, which are the first keys of each of the two level-1 KIR's. The pointers associated with these two keys are the relative sector addresses of the two level-1 KIR's, which happen to be HEX(0001) and HEX(0002). This 2nd-level KIR is stored in relative sector HEX(0003) of the Key File, and its contents are:

```
Keys:     000, 671, FFF, etc.
Pointers:  1,   2, FFF, etc.
```

The KDR is now updated. T0 = 2, to show that the index now has 2 levels. T2$ = HEX(0003), to show that the highest level index is located at relative sector HEX(0003).

Assuming that the next key to be added is 562, the search now proceeds as follows. 562 is compared to the entries in the level-2 index to see where it falls. It is greater than or equal to 000, but less than 671. Therefore it falls in the range 000 to 670. The pointer associated with 000 in the level-2 index is HEX(0001), and therefore the level-1 index stored in relative sector HEX(0001) is read. Then 572 is inserted in its proper place in the level-1 index, as before. The system knows when it has reached level 1 because it is counting down from T0 to 1 as each level is read and searched.

When the key 562 has been added, the Key File structure looks like this:

Sector	Level	Keys
1	1	000, 009, 147, 198, 276, 292, 562, 589, 591, FFF, etc.
2	1	671, 710, 730, 796, 809, 851, 903, 913, FFF, etc.
3	2	000, 671, FFF, etc.

As further keys are added, the KIR's on level 1 will again become full, and again the KIR must be split to provide room for all the keys. Let us assume that keys 401, 402, 403, 404, 405, 406, and 407 are added. The first six keys will cause sector 1 to be full, and the addition of 407 will make a split necessary. Relative sector HEX(0004) will be assigned to the new KIR, and the resulting structure will look like this:

Sector	Level	Keys
1	1	000, 009, 147, 198, 276, 292, 401, 402, FFF, etc.
2	1	671, 710, 730, 796, 809, 851, 903, 913 FFF, etc.
3	2	000, 403, 671, FFF, etc.
4	1	403, 404, 405, 406, 407, 562, 589, 591, FFF, etc.

Note that no new level has been added this time. In this example, there is room in the level-2 index to reference up to 15 level-1 KIE's. Therefore at least 15 x 8, or 120 records (and probably more, up to 225) can be accessed by a two-level index search.

Once the 2nd-level index is full, it is split in the same way the original KIR was split, and a third level is created, pointing to two 2nd-level KIR's, which in turn point to the first-level KIR's. The first-level KIR's always contain the pointers to the actual data records. As new levels are added, more superstructure is added, but the bulk of the Key File remains the same.

If for a given Key File there is an average of 10 KIE's per KIR, the number of records which can be accessed by a given number of levels of index is as follows:

INDEX LEVELS	NUMBER OF RECORDS
1	9
2	99
3	999
4	9,999
5	99,999
6	999,999
7	9,999,999
8	99,999,999

For the largest possible key (30 bytes), each KIR holds a maximum of 7 KIE's and a guaranteed average minimum of 4 KIE's. For such a file the maximum 8 levels of index access at least 65,535 records.

Perhaps the best illustration of the Key File structure for a large file could be obtained by running PRINT KEY FILE with an actual KFAM file. The structure can then be traced from the highest-level index sector (T2$, in KDR) down to the level 1 pointers to the actual data record.

The general procedure for locating a key in KFAM is as follows:

1) The number of index levels (T0) and the relative sector address of the highest-level index (T2$) are taken from the KDR.

2) The index sector (KIR) is read from disk.

3) A search of the KIR is made to locate the key. The search returns a pointer (T) to the key in the KIR which is equal to, or lower than, the key being searched.

4) The relative sector address of the KIR and the pointer to the KIE found (T) are stored in tables, T2$(T3) and VAL(STR(T$,T3)), defining the path taken to locate the particular key, where T3 is the current index level.

5) If the current index level is greater than 1, the sector address for the next lower level index is taken from the KIE found (T), and the process is repeated from Step 2 above for the next lower level.

6) If the current index level is 1, then the search is finished. T points to a KIE on level 1, and V indicates whether the key found is equal to or lower than the key being searched. Control is returned to the particular subroutine (FINDOLD, FINDNEW, DELETE, etc.).

The general procedure for inserting a key is as follows:

1) The proper position for the key is determined by the search procedure, above.

2) If the KIR is not full, the key and its associated record pointer are inserted at location T+1 in the KIR. All KIE's from location T+1 and up are moved up one position.

3) If the KIR is full, a new KIR is created on the same level as the old KIR. The KIE's are divided between the old KIR and the new KIR. The new key and its associated record pointer are inserted in proper sequential order in either the old KIR or the new KIR, depending on where the new key happens to fall. The next available sector address in the Key File is assigned to the new KIR.

4) If the split is not at the highest index level, the first key and the sector address of the new KIR are inserted in proper key sequence in the next highest level KIR (as determined by tables T2$() and T$). If the next highest level KIR is full, Step 3 is repeated at that level.

5) If the split is at the highest index level, a new level is created. A new KIR is created, with two KIE's. The first KIE contains the binary zero key and the relative sector address of the old KIR (formerly the highest level KIR). The second KIE contains the first key and sector address of the new KIR (created by the split). The next available sector in the Key File is assigned to this new highest level index. The KDR is updated (T0 and T2$) to reflect the new level.

When the KIR is split, it is not always divided equally unless an experienced programmer sets the moving bias. There is a reason for this. Consider keys which are being added sequentially. Again assume the first index sector is filled by 14 active KIE's and one dummy KIE.

```
000, 001, 002, 003, 004, 005, 006, 007, 008, 009, 010
011, 012, 013, 014
```

The next key added, 015, causes a split:

```
Old KIR:  000, 001, 002, 003, 004, 005, 006, 007, FFF, etc.
New KIR:  008, 009, 010, 011, 012, 013, 014, 015, FFF, etc.
Level 2:  000, 008, FFF, etc.
```

The next keys added, 016, 017, etc., are all added to the new KIR, eventually causing it to be split:

Sector	Level	Keys
1	1	000, 001, 002, 003, 004, 005, 006, 007, FFF, etc.
2	1	008, 009, 010, 011, 012, 013, 014, 015 FFF, etc.
3	2	000, 008, 016, FFF, etc.
4	1	016, 017, 018, 019, 020, 021, 022, 023, FFF, etc.

The process continues, always adding to the latest KIR and splitting it, leaving behind a residue of KIR's which are only half full. It should be clear in this case that if the split were 12/4 instead of 8/8, the process of indexing a sequential file would leave behind a residue of KIR's each containing 12 KIE's or 80% full. This would result in better utilization of Key File space and also tend to reduce the number of index levels required to access a given file.

But a 12/4 split would be disastrous if the keys were being added at random. There would be a greater probability of new keys being added to the KIR's already containing 12 entries because of the greater range of values represented. So the Key File could actually fall below 8 keys per sector, and a very inefficient skew distribution would be the result.

Therefore there is no particular split that is best in all cases. Because of this, a bias has been included in the system. The bias is a percentage of the maximum number of KIE's which, for a particular key size, can be contained in a KIR. When a KIR must be split, the current bias percentage is multiplied by the maximum number of KIE's per KIR to give the split, i.e., the number of KIE's which go into the new KIR. The bias may range from .2 to .8 and is set at .5 following the Open. An experienced programmer may set the bias after the file is opened.

On the basis of past experience, the bias should approach .2 as keys are added sequentially, should stay at .5 if keys are added in random order, and should be at .8 if records are added in backward key sequence. The bias affects disk space use and thus influences access times.

In REORGANIZE IN PLACE where it is known that keys will be added sequentially, the bias is set to .2 at the beginning. It is reset to .5 following the reorganization.

In KEY FILE CREATION the bias is set to .5 initially and reset to .5 when the program is finished because the order of keys added when initially creating the Key File could very well be different than the order of keys added at some later time (for example, sequential vs. random). The random hypothesis is always the "safest" to start with, unless experience proves differently.

Between the creation of the Key File and the reorganization (if any), the bias is allowed to fluctuate on the basis of how keys are added. It is stored in the KDR and preserved as a permanent record, i.e., not reset every time the program is reloaded.

In summary, for KFAM, there are two minor departures from the B-tree structure as described in Knuth: first, keys are duplicated in higher level indexes and second, a bias is available for the splitting of KIR's.

26.3 KEY FILE RECOVERY INFORMATION

KEY FILE RECOVERY allows reconstruction of a Key File in the event of its accidental destruction. In reconstructing the Key File, information saved by the CLOSE (WITH RECOVERY INFORMATION) or the WRITE RECOVERY INFORMATION subroutines in the next-to-last sector of the User File enables reconstruction of the Key File.

At the end of a User File are two sectors of "overhead". The last sector is a control sector written by the OPEN statement. In the next-to-last sector is a "trailer" record written during the INITIALIZE KFAM FILE utility. Two control bytes in this trailer record mark it as a trailer record for the 2200 system; however, the remaining bytes are ignored by the 2200 system logic. Some of these remaining bytes are used by KFAM to store recovery information. The information is stored each time the CLOSE subroutine is executed in the Shared or Exclusive access mode if the KFAM0107 module is in use or if a BUILD SUBROUTINE MODULE program was used and the CLOSE WITH RECOVERY INFO option was chosen.

The data saved by CLOSE in the next-to-last sector of the User File is the Key File's KDR record and is as follows:

Bytes	Contents
1-2	HEX(A0FD)
3-146	T$(3)48 as listed in Section 26.1 for the KDR.

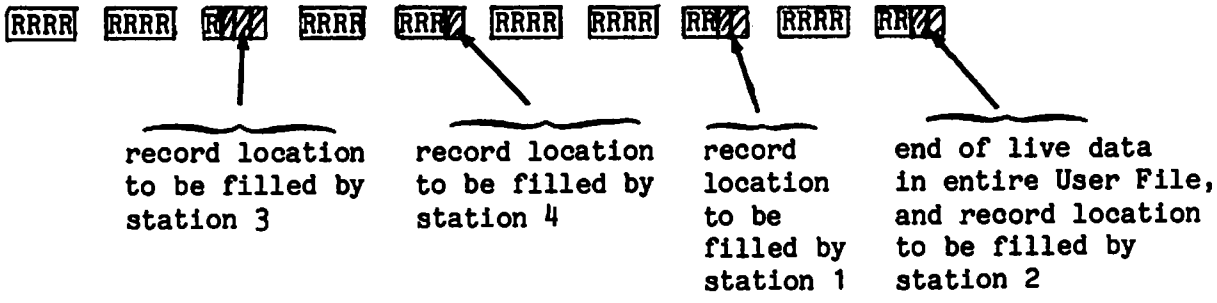
26.4 FINDNEW WITH BLOCKED FILES UNDER KFAM-7

FINDNEW always sets the Current Sector Address for the User File to the next available sector at the end of the live data in the User File. If records are blocked (type A, B, or C), it passes back the next record location.

Under KFAM-7, up to sixteen stations can have a KFAM file open simultaneously. When a station executes OPEN for a file, it is assigned a slot from one to sixteen in the KDR's access table. Associated with each slot in the access table is a relative sector location and, for blocked files, a record number within that sector. This sector location and record number always point to the last location in the User File assigned when a station, occupying that access table slot, executed a FINDNEW. If, after OPENing a blocked file, a station executes FINDNEW, the location passed to it (sector and record location within the sector) will be the next available location after the last location given to a station occupying the same access table slot. This new location will be the sector following the last sector of live data in the file only if a new block must be started.

In summary, when using blocked files with KFAM-7, whenever a new block must be used, FINDNEW assigns an entire block to a particular access table slot. That block then becomes the exclusive property of that slot in the access table for the purpose of FINDNEW. It can only be filled by FINDNEW's executed by a station occupying that slot. The result is that all record locations up to the end of live data in the User File may not be filled at any one time.

For blocked files under KFAM-7, the User File might look like this, for example,



where: R = record
▨ = unoccupied record locations

26.5 COMPATIBILITY BETWEEN KFAM-4 AND KFAM-7

KFAM-4 is upwards-compatible with KFAM-7, with the following exceptions:

a. Minimum key size is 2 bytes.

b. OPEN subroutine requires more parameters and has different error return codes.

c. Hog mode is now controlled by a $OPEN statement in the user's program.

d. Memory requirements for the full set of subroutines are greater, requiring about 2,000 extra bytes for the full set, or about 9K. This fact, however, is insignificant in view of the amount of memory saved by the global partitioning scheme.

e. A valid station number is required in variable S2.

f. A number of "R" variables are used in additon to Q, T, and V formerly reserved for system use.

26.6 COMPATIBILITY BETWEEN KFAM-5 AND KFAM-7

Data files created under KFAM-5 are valid as KFAM-7 data files, that is, KFAM-5 files are "media compatible" with KFAM-7. No conversion utility is needed.

The programming procedure, however, is slightly different. OPEN returns a Q$ value of S, unlike KFAM-5. Hog mode of the Key File's disk device has been dropped as an option. All disk hogging must be done by the user (although it is released either before or upon completion of a subroutine) with a $OPEN statement. The SET-UP subroutine (required for BUILD SUBROUTINE MODULE) has been dropped and is no longer required. On a return code of Q$="B", a $BREAK statement is recommended before attempting a retry. The memory usage inherent to the 2200MVP is different than the single user, single station 2200 CPU's, and special program statements are available for 2200MVP use (refer to the BASIC-2 Language Reference Manual for 2200VP/2200MVP systems). Variable V in the KDR is now always zero (0). Internal storage variable V0$(3)24 is now V0$(3)21 and has been changed internally.

CHAPTER 27

KFAM ADVANCED PROGRAMMING TECHNIQUES

27.1 FILES TOO LARGE FOR ONE DISK

A cataloged disk file must be wholly contained on one disk. If the User File is too large for one disk, it must be broken into two separate files. (Both files may have the same name, since they are on different disks.) Separate Key Files must be created, one for each User File. (If both Key Files are on the same disk, they may not have the same name.)

Perhaps the simplest scheme for splitting the User File is to determine a "cutoff" point. A key value is picked, somewhere in the middle, which will be the highest key in User File #1. Records with lower keys are stored in User File #1, and records with higher keys are stored in User File #2.

If each User File and its companion Key File are stored on the same disk, both User Files may have the same name, as may both Key Files. In that case, the same routines can be used to access both files by simply changing the disk address designation.

27.2 REUSING DELETED SPACE WITH FINDNEW(HERE)

Immediately following a DELETE, FINDNEW(HERE) may be used to insert a new record in the space just vacated by the deleted record. This function is useful for changing a key, but is not generally useful to reuse the deleted space because a new record is not generally available immediately following a DELETE.

The user may, however, store the pointer to the deleted record in a separate file for later use. The procedures are given below.

KFAM-7 does not check that FINDNEW(HERE) follows DELETE.

Under KFAM-7, the pointer to a deleted record may be saved as follows:

1. DELETE a record.

2. Test to make sure that Q$ = blank.

3. Save the contents of T4$ in some file or list external to KFAM. (See Section 26.1, "Internal Storage: User Partition".)

To re-use the space at some later time:

1. Move the saved record pointer to T4$. (See Note below.)

2. Use FINDNEW(HERE) with the new record key.

3. FINDNEW(HERE) will return with the Current Sector Address set to read the correct sector and Q = the record number within the sector.

> NOTE:
>
> If the file to be accessed is not the same as the file last accessed by a KFAM subroutine, move the saved record pointer to STR(T5$(i),1,3), where i = this file's KFAM I.D. Number. If not sure which file was last accessed, test T9 = KFAM I.D. Number last accessed.

27.3 MULTIPLE KEY FILES PER USER FILE

KFAM does not support multiple Key Files for a single User File. Though the Key File number provides a means of identifying different Key Files for a single User File, the subroutines and utility programs are designed for operations in which there is only one Key File per User File. The Programming Department of Wang Laboratories, Inc. does not support KFAM based file access systems that attempt to maintain multiple Key Files for a single User File. A protection scheme may be designed and implemented to allow such features as record protection using multiple Key Files, external to KFAM-7.

27.4 STATUS OF THE KEY DESCRIPTOR RECORD (KDR)

The fields which are of most interest to the user, T4$ (current pointer) and T7$ (current key), are stored internally. (See Section 26.1.)

There are legitimate reasons why a user may wish to change information in the KDR. One problem which is likely to occur is that the starting position of the key or the record length is wrong, causing a reorganization program to fail. These fields, which are critical in reorganizing, cannot really be checked prior to reorganizing. At the point of reorganizing, it is not generally feasible to re-create the Key File from the beginning. If such problems or similar problems occur, the contents of the KDR can be changed by the user via a very simple procedure:

```
S2 = (Station #)
SELECT (User File #, Key File #)
OPEN the file, Exclusive mode
Modify the appropriate KDR variable, e.g., T$()
CLOSE the file
```

This will read in the KDR, change it, and write it back on the disk. The KDR is always read by the OPEN subroutine and always written by the CLOSE subroutine in the Exclusive access mode.

27.5 FILE NAMES FOR THE KFAM UTILITIES

File (module) names for the KFAM-7 utilities are provided in Table 27-1.

Table 27-1. KFAM-7 Modules

UTILITY	MODULE NAME
Work File used to generate code and store error messages to be printed by KFAM2107. This is a data file of 15 sectors; all other files are program files.	KFAMWORK
KFAM-7 variables to be included in user program residing in user partition.	KFAM0007
KFAM-7 subroutines to be loaded into global partition.	KFAM0107
Start-up module for KFAM-7 Utility Programs.	START
ISS start-up operation.	ISS.001M
Menu and initial dialog	KFAM-7
INITIALIZE KFAM FILE	KFAM1007
KEY FILE CREATION UTILITY (also KEY FILE RECOVERY)	KFAM2007
Print duplicate keys, etc., using KFAMWORK.	KFAM2107
REORGANIZE IN PLACE:	
Start-up	KFAM3007
Generate Code	KFAM3107
Reorganize	KFAM3207
REALLOCATE KFAM FILE SPACE	KFAM4007
CONVERT from KFAM-3 to KFAM-7 or KFAM-4 to KFAM-7	KFAM5007
PRINT KEY FILE	KFAM6007
RESET ACCESS TABLE	KFAM7007
BUILD SUBROUTINE MODULE:	
Processing module	KFAM8007
Input file	KFAM0107
KEY FILE RECOVERY:	
Start-up (calls KFAM2007)	KFAM9007
Close files (called by all KFAM-7 utility programs)	KFAM9907
REORGANIZE SUBSYSTEM (standalone)	
ISS Reference File	ISS.REFB
Start-up, open files	KFAM3507
Generate code	KFAM3607
Reorganize, parts 1 and 2	KFAM3707
Part 3, close files	KFAM3907

CHAPTER 28

NON-KFAM FILE OPEN/END/CLOSE SUBROUTINES

28.1 OVERVIEW

Non-KFAM file Open/End/Close subroutines are designed for 2200MVP multistation disk environments and are included in the KFAM0107 global subroutine module.

Multistation disk files provide additional security features not available with regular disk files. The security features are accomplished by using only the multistation Open/End/Close subroutines on multistation files, instead of regular BASIC-2 Language DATASAVE DC OPEN, END, and CLOSE statements. Special information, namely an access table and Password, is maintained in a previously unused portion of the catalog trailer record by the multistation subroutines. Implementing a unique Password for a multistation file will allow only those who know the Password to access that file. In addition, the access table allows a station upon opening the file to have Exclusive (private) access to a file in the Exclusive mode or file access in one of three non-exclusive (public) modes including the Inquiry, Read Only, and Shared modes. The selected access mode is established each time the file is opened and is discontinued when the file is closed. Closing a file terminates file access, whereas opening begins file access for each station independent of other stations.

28.2 PASSWORD USE

When a file is first created by the Open subroutine, this is called an "open new" operation. Whether or not a Password will be required by all stations attempting access to that multistation file (later) is determined by the content of the argument (symbolic variable) P$ when the Open subroutine is called for the "open new" operation. If P$ contains blanks, a Password of blanks is required for any station to open that file. If P$ contains anything other than blanks, that value of P$ must be provided on any call of the Open subroutine for any user to access that file. When the "open new" operation has been successfully completed, the Password security feature is operable. Once set by the "open new", the Password cannot be changed. A Password may be implemented on a subsequent "open new" command to create a new file if not implemented when the file was previously created on an "open new".

28.3 CONVERTING TO MULTISTATION FILES

The conversion from a regular disk data file to a multistation disk data file occurs when that file is opened using the Open subroutine. Thus, by accessing the file, it is converted to a multistation file automatically; however, for the file to remain a multistation file, the Open/End/Close subroutines always must be used instead of the DATASAVE DC statements.

> CAUTION:
>
> The following BASIC-2 instructions destroy the access table (and Password) necessary for multistation files: DATASAVE DC END, MOVE TO, and MOVE. Instead, use the END subroutine and Copy/Verify ISS Utility or COPY statement respectively.

28.4 OPEN SUBROUTINE (DEFFN' 217)

The Open subroutine, by virtue of its arguments, allows a multistation file to be opened, the access mode defined, and a password (if any) used. When creating a new file, this is called an "open new." When accessing an existing (old) file, this is called "open old." In addition, when accessing an existing file the access mode may be changed; this is called a "re-open old." The Open subroutine replaces the BASIC-2 Language statement DATASAVE DC OPEN and similarly allocates and reserves file sectors on an "open new" operation.

Access Modes

The four access modes include Inquiry, Read Only, Shared, and Exclusive. In the Exclusive mode, only the station with Exclusive access may open the file (private access). However, in order for that user to have Exclusive access, no other station can have that file open. The Exclusive access mode is required for an "open new".

A file may be opened in the Inquiry mode if that file is not currently open in the Exclusive mode to another station. Conversely, a file open in the Inquiry mode keeps stations requesting the Exclusive mode from opening that file.

A file may be opened in the Read Only mode if that file is not currently open in the Shared or Exclusive mode to another station. Conversely, a file open in the Read Only mode keeps stations requesting the Exclusive or Shared Modes from opening that file.

With the Shared mode requested, that file will be opened successfully if that file is not open in the Read Only or Exclusive modes. Conversely, a file open in the Shared mode keeps stations from opening that file requesting the Read Only and Exclusive modes.

A file may be opened in the Exclusive mode if no other stations are accessing that file. Once a file is opened in the Exclusive mode, no other stations can access that file until it is closed.

Disk Hog Mode

With the availability of Exclusive file access, the need for disk Hog mode is significantly reduced. Only in cases where more than one file must be opened with Exclusive access on one disk is the use of Hog mode usually necessary. One advantage of Hog mode over Exclusive access is that with Hog mode, there is no wait for other stations to close files for which Exclusive access is needed, which is an important consideration if more than one file must be opened with Exclusive access. (Exclusive access requires that no other stations have that file open in order for the Open subroutine to be successful.) Hog mode is controlled by the BASIC-2 statements $OPEN and $CLOSE. Hog mode is released upon executing a subroutine or if the program is hogged, therefore only the $OPEN statement applies.

GOSUB' 217 Argument Format

> NOTE:
>
> The file number (F) must be assigned to its disk address prior to calling the Open subroutine (use a SELECT statement). The station number, S2, must coincide with (argument) symbolic variable C.

Transfer to the Open subroutine occurs via the statement:

```
GOSUB' 217(F$,F,C,S,A,P$,A1$,0)
```

where the symbolic (dummy) variables denoting each argument are as follows:

F$ indicates the file name, which on an "open new" command must be unique to that disk. On any "open old", it must be identical to the previously assigned file name.

F contains the file (device) number, from 0 to 15.

C contains the station (partition) number, from 1-48, e.g., common variable S2.

S determines the type of open command. If S is greater than zero, this indicates an "open new" command; also, S equals the number of sectors to be allocated for this file. If S=0, this indicates an "open old" command. If S=-1, this indicates a "re-open old" command during which the access mode is changed.

A determines the access mode for this file. A=1 indicates Inquiry mode. A=2 indicates Read Only mode. A=3 indicates Shared mode. A=4 indicates Exclusive mode. For an "open new", A must equal 4.

P$ contains a password, if one is required. Password content is determined solely by that file's "open new".

A1$ contains the disk address in the xyy form (not used).

0 (zero) is a required argument.

GOSUB' 217 Return Codes

Q$ contains the return code as summarized below in Table 28-1.

Table 28-1. Non-KFAM File OPEN Return Codes

VALUE Q$	INDICATES	CAUSE AND RECOVERY
blank	successful open	Continue
A	access mode conflict	Retry, or wait and retry. If abnormal delay, check if value of argument A is correct. Check if correct file was accessed (values F$,F,C). Check if that file was accidentally left open (use File Status Report utility).
D	file not found or the open command conflicts with file disposition, e.g., "open new" issued to existing file, "open old" issued to a file already open, or "re-open old" issued to closed file.	Check if value of argument S is correct. Check if correct file was accessed (values F$,F,C). On "open old", check if that file was accidentally left open (use ISS File Status Report utility).
S	insufficient disk space to complete "open new" allocation.	Retry after reducing value of S if acceptable, or use different disk by changing F. Otherwise, use Free Unused Sectors Disk subroutine to create more disk space and retry "open new" with S as is.
P	password conflict	Check value of P$. Retry with correct file password.

28.5 END SUBROUTINE (DEFFN' 218)

The End subroutine performs the function of the BASIC-2 statement DATA SAVE DC END for multistation files. As with the DATA SAVE DC END, upon calling the End subroutine, the Current Sector Address must be at the location required for the end-of-file trailer record. Use of the DATASAVE DC END on a multistation file results in destruction of the access table necessary for multistation files.

Transfer to the End subroutine is via the statement:

```
GOSUB' 218 (F$,F,A1$,0)
```

where the symbolic variables denoting the arguments are as follows:

F$ is the file name.
F is file (device) number, from 0 to 15.
A1$ is the disk address in the xyy form (not used).
0 (zero) is required.

There are no return codes.

28.6 CLOSE SUBROUTINE (DEFFN'219)

Closing a file promptly with multistation operation is usually important because of possible access mode conflicts. Once access to a file is no longer needed, that file should be immediately closed, especially if opened with Exclusive access.

To retain a file for later processing, the file may be left open if needed. There is no limit as to how many files can be open for one station because station/file status is maintained in the file's catalog trailer record, and not by each partition. Because the access table resides on non-volatile storage of disk, memory may be cleared or power switched OFF, and the file will remain open to that station. This type of action, of course, is not recommended. Files should not accidentally be left open.

Transfer to the Close subroutine occurs via the following statement:

```
GOSUB' 219 (F$,F,C,A1$,0)
```

where the symbolic variables denoting the arguments are as follows:

F$ is the file name.
F is the file (device) number.
C is the station (partition) number, from 1 to 48.
A1$ is the disk address in the xyy form (not used).
0 (zero) is required.

There are no return codes.

PART IV

THE ISS SCREEN/DISK SUBROUTINES

CHAPTER 29 - OVERVIEW OF THE SCREEN/DISK SUBROUTINES

29.1 INTRODUCTION

The ISS Screen/Disk Subroutines comprise a library of marked subroutines designed to eliminate the repetitious, detailed programming tasks otherwise required of an application programmer. These marked subroutines -- known as the ISS Screen/Disk subroutines -- provide a simple interface between application programs and a wide range of potentially complex disk-related and operator-related tasks.

There are three groups of Screen/Disk subroutines: the DISK subroutines, the SCREEN subroutines, and the TRANSLATION TABLES subroutines. The SCREEN subroutines perform various tasks related to the interaction between operator and station, whereas the DISK subroutines perform tasks related to station and disk interaction. The TRANSLATION TABLES subroutines initialize 256-byte arrays with the proper hex codes for four standard character code translations; these arrays are designed for use with the BASIC-2 statement $TRAN.

The subroutines selected may be specified either as "global" or "non-global" (local). In either case, the subroutines chosen are loaded into memory (the partition in use) and saved to disk.

If "global" is specified, two program files are output to a user-specified disk address with user-specified file names. One program file contains Dimension (DIM) statements for certain variables which must be incorporated into the user's application program; the other program file contains program text consisting of the selected subroutines to which the user adds a DEFFN @PART statement in order for the subroutines to be referenced (later) by multiple stations as a global partition.

If "non-global" (local) is specified, only one program file is output, and it contains both Dimension statements and subroutines which are both incorporated into the user's application program. A disk address and file name are specified by the user.

Symbolic Variables and Arguments

The DEFFN' statement which marks each Screen/Disk subroutine may require certain parameter values to be passed from the GOSUB' statement which calls it. Parameter values passed are assigned to certain variables within the subroutine. If parameters (arguments) are required, "symbolic variables" (i.e., "dummy variables") listed in this manual denote each argument required following the appropriate GOSUB' statement. Symbolic variables are not the actual variables required in an argument list. Instead, symbolic variables indicate whether a numeric expression or alphanumeric expression is required in place of the symbolic variable.

If a symbolic (dummy) variable's name is numeric, a numeric expression such as a number or a user-defined numeric variable is required in its place. If a symbolic variable's name is alphanumeric, an alphanumeric expression such as an alphanumeric literal (within quotation marks) or user-defined alphanumeric variable is required in its place. This convention attempts to ensure that an alphanumeric expression (argument) is not assigned to a numeric variable in a subroutine, and vice versa.

Generally, the letter chosen for a symbolic variable's name is the first letter of the associated parameter's name, e.g., L represents Length.

Reserved ISS Variables and DEFFN' Numbers

All variables (scalar and array, alpha and numeric) in the range Q through W are reserved for use by ISS. Such variables should be handled as "read only" variables by the user's program, unless the description of a specific subroutine states otherwise (e.g., default values for Data Entry, a Screen subroutine). Similarly, all DEFFN' statements in the range 200-255 are reserved solely for ISS subroutines. While individual items within these ranges may not be used on a given release of ISS, in supporting ISS it is assumed that no variables or DEFFN' subroutines in these ranges are used for purposes unrelated to the subroutines.

All subroutines are compatible with one another in regard to variable usage. However, all Translation Table subroutines load the same array variable. Also, when calling the same subroutine more than once, before calling the subroutine the second (or next) time, any information returned from the previous call should be either processed or equated to a user-defined variable.

If a subroutine argument specifies a disk file number, a disk address must be selected for that file number prior to calling the subroutine. File numbers are selected by executing a SELECT statement such as 50 SELECT #3/310.

29.2 CHOOSING AND SAVING SCREEN/DISK SUBROUTINES

All Screen/Disk subroutines may be loaded simultaneously. Because none of the subroutines destructively overlaps another, all subroutines may be used in a single program, if desired. Statements are numbered within the range 71-90 and 6000-9899, where DIM (Dimension) statements are located on statement lines 71-90 and DEFFN' subroutine program text is located on lines 6000-9899. Program lines associated with the menus are located outside these ranges.

Following ISS start-up operation (see Chapter 3), the SCREEN ROUTINES menu appears. Each group of subroutines has its own menu; the menu's name appears within parentheses below:

1. Screen subroutines (SCREEN ROUTINES)
2. Disk subroutines (DISK ROUTINES)
3. Translation Table subroutines (TRANSLATION TABLES)

Choosing the Desired Subroutines

In reply to the menu currently displayed, the user chooses the subroutines to be output by touching the corresponding S.F. Keys. As each subroutine is chosen, an asterisk (*) appears to the left of the subroutine chosen and also to the left of any subroutines automatically included with the chosen subroutine.

In addition to the S.F. Keys available to choose the desired subroutines, the following S.F. Keys are available:

1. To obtain the next menu of subroutines, touch S.F. Key '16. After touching S.F. Key '16, if the Screen subroutines menu was displayed, the Disk subroutines menu appears; if the Disk subroutines menu was displayed, the Translation Table subroutines menu appears; if the Translation Table subroutines menu was displayed, the Screen subroutines menu reappears. In this manner, the user can obtain the next menu and choose the subroutines desired from that menu, obtain the next menu and choose the subroutines desired from that menu, etc.

   After choosing the subroutines desired, S.F. Key '16 should be touched to allow the user to visually verify the subroutines chosen from the three subroutine menus.

2. To erase all subroutines chosen (indicated by asterisks) from all menus, touch S.F. Key '18. Because S.F. Key '18 erases all asterisks, the user should again choose all the subroutines desired from the three menus.

Saving the Subroutines Chosen to Disk

After visually verifying that the correct subroutines have been chosen, the user has two options:

1. To save the chosen subroutines for subsequent non-global (local) use, touch S.F. Key '26. The chosen subroutines are loaded and the following prompts appear:

ENTER OUTPUT ADDRESS - requests entry of the disk device address (xyy form) where the chosen subroutines are to be saved. Valid ISS disk addresses are displayed.

MOUNT OUTPUT DISK, KEY RETURN(EXEC) TO CONTINUE?-requests mounting of the disk(ette) where the chosen subroutines are to be saved (the disk address just entered). Touch RETURN when ready.

ENTER FILE NAME - requests entry of the file name to be assigned to the program file to contain the selected subroutines.

"SAVING ROUTINES" and "STOP ROUTINES SAVED" appear. To obtain the ISS start-up prompt "ENTER DESIRED FUNCTION", from which the Applications menu may be obtained, touch S.F. Key '31.

2. To save the chosen subroutines for subsequent global use, touch S.F. Key '28. The chosen subroutines are loaded and the following prompts appear:

   ENTER OUTPUT ADDRESS - requests entry of the disk device address (xyy form) where the chosen subroutines and Dimension statements are to be individually saved as program files. Valid ISS disk addresses are displayed.

   MOUNT OUTPUT DISK, KEY RETURN(EXEC) TO CONTINUE?- requests mounting of the disk(ette) where the chosen subroutines are to be saved (the disk address just entered). Touch RETURN when ready.

   ENTER FILE NAME FOR 'VARIABLES' - requests entry of the file name to be assigned to the program file to contain the Dimension statements for certain variables. This program file's contents are incorporated into the user's application program.

   ENTER FILE NAME FOR 'TEXT' - requests entry of the file name to be assigned to the program file to contain the chosen subroutines (program text). This program file's contents (later) should be loaded, a DEFFN @PART statement added to it, and it should then be resaved for subsequent use as a global program file. (For the automatic bootstrap feature to be used, the global program file should be resaved to the 2200MVP Operating System Diskette.)

   "SAVING ROUTINES" and "STOP ROUTINES SAVED" appear. To obtain the ISS start-up prompt "ENTER DESIRED FUNCTION", from which the Applications menu may be obtained, touch S.F. Key '31.

Estimating Partition Memory Size Requirements

In order to load a program file containing all ISS Screen/Disk subroutines, a 6.75K partition is necessary; however, in order to load and run a program file containing all ISS Screen/Disk subrotines, a 8.25K partition is necessary. These partition requirements are the same for both global and non-global subroutines. The variables required in the global output require .5K to load and a 2.0K partition to load and run if all Screen/Disk subroutines are chosen. Because it is not likely that all Screen/Disk subroutines will be chosen, use of the END statement or PRINTSPACE (SPACE function) allows the user to determine amount of unused (free) memory, and the amount of memory actually necessary.

CHAPTER 30 - SCREEN SUBROUTINES

30.1 INTRODUCTION

Screen subroutines, in general, control CRT screen messages and subsequent keyboard entry when incorporated into an application program. Each subroutine has a standard set of functions associated with it, such as cursor positioning, numeric or alphanumeric entry, checking of keyed input, entry of the date in Gregorian or Julian form, and similar operator-interactive functions.

Consistent implementation of Screen subroutines within a program (or set of related programs) results in operation that also is consistent. Simple and consistent operator/machine interaction is certainly a worthy goal, because it directly affects operator productivity and accuracy.

30.2 DATA ENTRY (DEFFN' 200)

The Data Entry subroutine accepts a keyboard entry and determines if it is acceptable for either numeric or alphanumeric input. Using the LINPUT statement, entries can be checked for (1) values either within a specified numeric range or alphanumeric limits, (2) the length of an alphanumeric entry, and (3) the length to the left of, and to the right of, the decimal place for a numeric entry. With numeric entry, all digits (0-9), minus sign, a decimal point, and Exponential characters including the uppercase letter E are valid entries.

A prompt can be displayed on line 1 and operator modifiable or unmodifiable defaults can be implemented to reduce the required number of operator keystrokes. With operator modifiable defaults, Edit mode is activated allowing nondestructive editing of the displayed default.

During the call to the Data Entry subroutine, any Special Function (S.F.) Key is a valid operator response if (1) the user's application program provides the corresponding DEFFN' statement followed by either a RETURN or a RETURN CLEAR statement and (2) Edit mode is not active.

Checking Features

After keying RETURN, checks are performed on the entire response including the following:

1. With a numeric entry, does the response conform to the minimum and maximum (range check) values specified, if any? (See arguments L$ and H$ below.)

2. If numeric entry, is the number of digits to the left, and to the right, of the decimal point within the maximum length limit specified for each? (See arguments L1 and R1 below.)

3. If alphanumeric entry, is the length within the maximum limit specified? (See argument L1 below.)

4. If alphanumeric entry, does the response conform to the alphanumeric limits specified? (See arguments L$ and H$ below.)

If the tests find the entry acceptable, a valid response is returned in scalar variable Q9 for numeric entry, or alpha-variable Q6$ for alphanumeric entry. If defaults are to be used, the default value must be equated to the appropriate variable prior to each call of the Data Entry subroutine.

With alphanumeric input, if an entry is rejected based on the arguments specified, "RE-ENTER" is displayed on line 3, the audio alarm sounds, and the subroutine is readied to accept data. With numeric input where the entry falls outside of the specified range or decimal point/digit length, "RE-ENTER L$>=ENTRY>=H$ (ddd.dd)" is displayed to indicate the entry failed the numeric range or length check, where the (ddd.dd) indicates the entry mask according to arguments L1 and R1.

GOSUB' 200 Argument Format

Transfer to the Data Entry subroutine occurs via the statement:

```
GOSUB' 200 (L$, H$, L1, R1, P$, T)
```

Numeric Input Arguments

Numeric input (symbolic variable) arguments are described below:

Range Check - L$ contains the lowest acceptable numeric entry for the field. Since L$ is itself alphanumeric, it must always be expressed as an alphanumeric string, e.g., L$ = "-99.99".

H$ contains the highest acceptable numeric entry for the field and must be expressed as an alphanumeric string, e.g., H$= "+99 99".

If L$ and H$ both contain blanks, no range check is performed.

Length Check - L1 equals the maximum number of characters to the left of the decimal place.

R1 equals the maximum number of characters to the right of the decimal place. L1 plus R1 cannot exceed 19.

If L1 and R1 both equal zero, no mask is displayed and a length test is not performed, but keystrokes are accepted.

Prompt - P$ is the alphanumeric prompt (to be displayed on line 1 of the CRT).

Type of Entry - T determines if a default value is used, as follows.

If T = -1, numeric input with an operator modifiable default value occurs. The default value contained in alpha-variable Q6$ is displayed and may be modified if desired by the operator. Edit mode is activated.

If T=1 numeric input occurs without default.

If T=0 numeric input with default occurs without display. The default value contained in scalar variable Q9 is used only if the operator accepts the default value by keying RETURN before entering any other characters.

Return - A valid operator reply is contained in scalar variable Q9.

Alphanumeric Input Arguments

Alphanumeric input arguments are described below:

Limit Check - L$ contains the lowest acceptable alphanumeric string value of the entry.
H$ contains the highest acceptable alphanumeric string value of the entry. If L$ and H$ are blank, no limits check occurs.

Length Check - L1 equals the maximum number of characters for the field, up to 61 characters.
R1 equals zero (0).

Type Of Entry - T must either equal 2, to indicate alphanumeric input without default, or equal 3, to indicate alphanumeric input with a default. If T = 3, the default characters contained in alpha-variable Q6$ are displayed and may be modified if desired by the operator. Edit mode is activated if T = 3.

Prompt - P$ is the alphanumeric prompt.

Return - A valid operator reply is contained in alpha-variable Q6$.

30.3 DATE ROUTINES (DEFFN' 220,221,222,223,224,225)

The "DATE ROUTINES" consist of a group of independently accessible subroutines which facilitate the entry and use of dates. Dates may assume two forms. These are known as the "Gregorian" and "Julian" forms, respectively. Gregorian form is alphanumeric MM/DD/YY

Where: YY is the 2 low order digits of the year.
MM is the number of the month such that $1 \geq MM \geq 12$.
DD is the day of the month $1 \geq DD \geq 31$.

Julian form is numeric YYDDD.

Where: YY is the 2 low order digits of the year.
DDD is number days since the beginning of YY counting January 1 as 1.

A Julian date is in proper form if

$YY \geq 0$ and
$1 \geq DDD \geq 365$ whenever YY specifies a non-leap year,

or $1 \geq DDD \geq 366$ whenever YY specifies a leap year.

A Julian date must be in proper form to be correctly converted to Gregorian form by any of the subroutines.

All the routines are designed to automatically account for leap years.

Enter Date - Gregorian Form

This subroutine provides for keyboard entry of a Gregorian date. It returns the entered date in Gregorian and Julian form. A prompt must be specified. The entered date is displayed in Gregorian and Julian form for operator verification before the subroutine is exited.

The subroutine is entered via

```
GOSUB' 220 (P$)
```

Where: P$ is the prompt, 64 characters maximum.

The prompt is displayed on line 1, and the cursor appears on line 2.

The slashes (/) in the date must be entered along with the appropriate digits (MM/DD/YY form), although leading zeroes need not be entered. If MM or DD assume values outside their valid ranges, entry is requested again.

Otherwise, the message IS DATE OK (Y/N) appears on line 2 with the entered date in its Gregorian and Julian forms. If N is entered, entry is requested again. If Y is entered, the Gregorian date is returned in alpha-variable U9$ and the Julian in scalar variable U9; the subroutine is exited.

Convert Date - Gregorian to Julian

This subroutine converts a date from Gregorian to Julian format. It is entered via

```
GOSUB' 221 (G$)
```

Where: G$ is the Gregorian date to be converted.

The routine returns alpha-variable U9$ with the Gregorian date and scalar variable U9 with the Julian equivalent of G$. If G$ could not be converted because the values of MM or DD were outside the valid range, alpha-variable Q6$ is returned as "E".

Enter Date - Julian Form

This subroutine provides for keyboard entry of a Julian date (YYDDD form). A prompt must be specified. The entered date is converted to its proper form (via GOSUB'224). The date is displayed in Gregorian and Julian form for operator verification.

The subroutine is entered via

```
GOSUB' 222 (P$)
```

Where: P$ is the prompt, 64 characters maximum.

The prompt is displayed on line 1, and the cursor appears on line 2. No check is performed to ensure the proper form of the entered Julian date.

The message IS DATE OK (Y/N) appears on line 2. with the entered date in its Gregorian and Julian forms. If a Julian date was entered which was not in proper form, the Gregorian date is incorrect. If N is entered, entry is requested again. If Y is entered, the Gregorian date is returned in alpha-variable U9$ and the Julian in scalar variable U9; the subroutine is exited.

Convert Date - Julian to Gregorian

This subroutine converts a date from Julian to Gregorian form. The date specified as an argument is converted to its proper form (via GOSUB'224). It is entered via

```
GOSUB' 223 (J)
```

Where: J is the Julian date to be converted.

The routine returns alpha-variable U9$ with the Gregorian equivalent of J and scalar variable U9 with the entered Julian date. No check is performed on J. A Julian date not in proper form will produce a Gregorian date with MM or DD outside the valid range.

Convert Julian Date to Proper Form

This subroutine converts any 5-digit Julian date to a Julian date in proper form, i.e., the number of days specified must be valid for the specified year and is converted if invalid (see examples below). It is entered via

```
GOSUB' 224 (J)
```

Where: J is a Julian date.

The subroutine returns the entered date in Q9 in proper form.

For example:

72367 is returned as 73001
71733 is returned as 73002

Calculate Days Between Two Dates

This subroutine calculates the number of days between two Julian dates. It is entered via

GOSUB' 225 (J1, J2)

Where:

J1 is the earlier date.

J2 is the later date.

U3 is returned equal to the number of days between J1 and J2.

For example:

If J1 = 75004 and J2 = 75009,

then U3 is returned as 5.

If J1 = 71360 and J2 = 72060, then U3 is returned as 65.

30.4 POSITION CURSOR (DEFFN' 248)

The Position Cursor subroutine moves the cursor to any location on a 16 x 64 or 24 x 80 display screen. Also, it can optionally erase the characters to the right of the new cursor position on the same line as well as entire lines below it. Position Cursor automatically determines the type of CRT used. Unlike the PRINT AT function which erases the specified number of characters, the Position Cursor subroutine erases the specified number of lines. The PRINT AT function, if desired, may be used instead of Position Cursor. Usually a PRINT or LINPUT statement, or screen subroutine equivalent, follows the Positon Cursor call.

Transfer to the Position Cursor subroutine occurs via the statement:

GOSUB' 248 (R,C,E)

where:

R = row (i.e., line number minus one), relative to zero
C = column, relative to zero
E = number of lines to erase

The cursor is moved to the position specified by the R,C argument relative to zero (0), e.g., R=0 and C=2 move the cursor to the top-most line, position 3. The absolute value of E determines if lines are erased and is identical for the two types of available display screens.

If E=0 (zero), no characters are erased. If E=-1 or E=1, only characters in the same line (row) to the right of the cursor's new position are erased. If E is less than minus one (-1) or greater than one, characters to the right of the cursor's new position (on that line) are erased, as are all lines below, to the value of E-1 lines for E values greater than one, or the absolute value of E minus one for E values less than minus one. Also, if E=-9E99 or E=9E99, the entire screen is cleared.

30.5 OPERATOR WAIT (DEFFN' 254)

This subroutine displays the message "KEY RETURN(EXEC) TO RESUME?" on line 2. Execution is halted on an INPUT instruction until RETURN is touched. Up to 64 entered characters are returned in alpha-variable Q6$.

Transfer to the subroutine is via the statement:

```
GOSUB' 254
```

30.6 RE-ENTER (DEFFN' 255)

If the Data Entry subroutine is selected, the RE-ENTER subroutine may be used.

The RE-ENTER subroutine displays the word "RE-ENTER" on line 3 of the screen when called. It is used to signal the operator of an entry error.

Transfer to the RE-ENTER subroutine is via the statement:

```
GOSUB' 255
```

30.7 PRINT ROUTINE (DEFFN' 242)

The Print Routine subroutine allows a specified character to be printed a specified number of times.

Transfer to the Print Routine subroutine is via the statement:

```
GOSUB' 242 (N, C$)
```

where: N = the number of times to print the character.
C$ = the character to be printed.

CHAPTER 31 - DISK SUBROUTINES

31.1 INTRODUCTION

The Disk subroutines are DEFFN' subroutines that, in general, provide specialized disk file maintenance functions. Each subroutine has a unique set of standard functions associated with it, but are not to be confused with KFAM subroutines. The Disk subroutines described in this chapter perform such functions as return the names of files on a disk in index sector sequence, return the status of a disk file, allocate the maximum possible amount of disk space for a new file and create it, de-allocate unused sectors in an existing file, as well as opening special data files to be used for output or input purposes and subsequently closing them.

Multistation Open/End/Close subroutines (i.e., Multiplex subroutines) are discussed separately in the next chapter, although they are selectable from the Disk subroutine menu.

31.2 SEARCH INDEX (DEFFN' 229)

The Search Index subroutine searches a disk catalog index for a specified file name. It returns the status of the file as active, scratched or nonexistent, as well as indicating whether the file is a data or program file. It is recommended that the BASIC-2 statement LIMITS be used instead of Search Index for efficiency, however, please note that Search Index returns the information in alpha-variable R2$ which is not returned by the LIMITS statement.

The subroutine is called by:

```
SELECT #F/xyy
GOSUB' 229 (F, N$)
```

where: xyy is the disk device address.
F is the File Number.
N$ is the file name.

The scalar variable R is used as a return code to indicate one of the following:

```
 2 - active data file
 1 - active program file
 0 - file does not exist
-1 - scratched program file
-2 - scratched data file
```

In addition, alpha-variable R2$ returns the file status code.

```
R2$ = HEX(10) the file is active.
R2$ = HEX(11) the file is scratched.
R2$ = HEX(00) the named file does not exist.
```

31.3 ALLOCATE DATA FILE SPACE (DEFFN' 228)

This subroutine opens a data file on any selected disk and allocates to it the available sectors between the current end of cataloged files and the end of the cataloged area. It checks the catalog index to ensure the uniqueness of the file name; it allows a minimum acceptable file size to be specified.

This subroutine is designed to be a counterpart to Free Unused Sectors.

The subroutine is called by

```
SELECT @F/xyy
GOSUB' 228 (F,N$,S)
```

where: xyy is the disk device address.
F is the File Number.
N$ is the name of the new file.
S is the minimum acceptable number of sectors for the file; if S is less than 0, the absolute value of S is allocated.

There are three conditions sufficient to prevent the file from being opened. In the sequence of their evaluation they, and their return codes, are:

a. If the file name is the same as an indexed scratched file, the return code R2$ is set to 3.

b. If the file name is the same as an indexed active file, the return code R2$ is set to 2.

c. If there are insufficient sectors in the catalog area, beyond the current end, to open the specified minimum file, the return code R2$ is set to 1.

If none of these conditions occurs, the file is opened and the return code R2$ is set to 0.

31.4 FREE UNUSED SECTORS (DEFFN' 227)

This subroutine examines a selected file in a disk catalog area. It de-allocates those sectors between the end of the file and the DATASAVE DC END trailer. It repositions the end of file control sector. The de-allocation may be restricted by specifying that a minimum number of extra sectors be maintained in the file (reserved for file additions).

The file must have been ended with a DATASAVE DC END statement. If this subroutine is executed on a file which lacks a DATASAVE DC END trailer, the file is destroyed.

This subroutine is designed as a counterpart to Allocate Data File Space.

The subroutine is called by

```
SELECT #F/xyy
GOSUB' 227 (F, N$, S1)
```

where: xyy is the disk device address.
F is the File Number.
N$ is the name of the file to be examined.
S1 is the number of extra sectors to be maintained in the file.

There are two independent conditions under which the file will not be altered. In the sequence of their evaluation, they, and their return codes, are:

a. If the file does not exist, the return code R2$ is set to 3.

b. If the number of extra sectors found in the file is less than or equal to the number of extra sectors to be maintained in the file, the subroutine returns 1 in R2$.

If none of the above conditions occurs, the file is altered and the subroutine returns 0 in R2$.

Note that if the file is the last file in the catalog area, Free Unused Sectors updates the end of catalog, as well as the end of file.

31.5 LIMITS NEXT (DEFFN' 226)

The Limits Next subroutine returns the names of files on a disk in index sector sequence, the same order provided by the LIST DC statement. Also returned for each file is its file status and whether it is a data or program file. For each call the next file name in sequence and its corresponding status is returned.

Transfer to the Limits Next subroutine occurs via the statement:

```
GOSUB' 226 (F,N$)
```

where: F = The disk device number in the XYY form.

N$ = The file name the sequence will begin at. If N$=HEX(0000000000000000), the scan will begin with the first file in sequence.

After return, this routine contains the following values:

R9$ = The name of the next file in sequence
If R9$ equals HEX (0000000000000000),
the end of file sequence has been encountered.

R = The file status of the file name returned in R9$, where:

- 2 - indicates active data file.
- 1 - indicates active program file.
- 0 - indicates file does not exist (occurs at end of index).
- -1 - indicates scratched program file.
- -2 - indicates scratched data file.

The initial call provides F and N$; thereafter on return the program should test for R=0. If R does not equal 0, of course, the program loop continues calling GOSUB' 226.

31.6 OPEN/CLOSE OUTPUT (DEFFN' 240, 241)

These subroutines open for output, and subsequently close, disk data files which utilize special header and trailer information. In addition to satisfying the file open and close requirements for disk catalog operation, they produce single sector software header and trailer records with the following fields:

FIELD	TYPE	FIELD LENGTH	DISK LENGTH	CONTENTS
1	Alphanumeric	3	4	HDR-indicates header EOF-indicates end of file EOR-indicates end of volume
2	Alphanumeric	8	9	file name
3	numeric	8	9	creation date (Julian format)
4	numeric	8	9	number of days to retain file (the "retention period")
5	numeric	8	9	volume number

Based on the data in the header and trailer records, these subroutines enforce certain system standards. For example, when a file is opened for output, a life span in days is specified for it. The file, then, cannot be opened for output again until this life span has expired.

Open Output

The subroutine is called by

```
SELECT #F/xyy
GOSUB' 240 (F,N$,D,V)
```

where: xyy is the disk device address.
F is the file number.
N$ is the name of the file to be opened.
D is the number of days the file is to be preserved (the "retention cycle")
V is the volume number of the file.

The subroutine displays the message MOUNT DISK TO CONTAIN VOL. XX OF FILE (FILE NAME) UNIT X. After the specified disk is mounted, the catalog index is searched for the file name.

If the file is not listed in the disk index, it is opened using the technique of the Allocate Data File Space subroutine. Up to three files may be open.

If the file is indexed but scratched, the scratched file is reopened as an active file.

If the file is indexed and active and the retention period has expired, the file is reopened.

Regardless of which one of the above conditions is found, the subroutine writes the software header record in the first available file sector. The Julian date is obtained from Q1. Control returns to the application program with the read/write head at the first available file sector after the software header.

If the file name is in the index, active, but the retention period has not expired, the message RETENTION CYCLE NOT EXPIRED appears together with the mount message. If there is insufficient space to open a file, the message INSUFFICIENT SPACE appears together with the mount message.

> NOTE:
>
> Keying X and RETURN in response to the mount message causes any file with the same name to be reopened.

Close Output

The subroutine is called by

```
SELECT #F/xyy
GOSUB' 241 (F, T$)
```

where: xyy is the disk device address.
F is the file number.
T$ is the software trailer indicator
"EOF" for end of file, or
"EOR" for end of volume.

The subroutine writes the software trailer followed by the hardware (DATASAVE DC END) trailer.

If the file is the last file in the catalog area, the techniques of the Free Unused Sectors subroutine are employed to return the unused sectors to the available disk catalog area.

The file is closed and a message requests removal of the disk.

If T$ is set to "EOF", control is returned to the application program. If T$ is set to "EOR", the volume counter is incremented for the next software header, and the Open Output subroutine is called again.

31.7 OPEN/CLOSE INPUT (DEFFN' 250,251)

These subroutines open for input and subsequently close disk data files which utilize special header and trailer information. They are designed to work in conjunction with the Open/Close Output subroutines and depend upon properly structured software headers and trailers.

Open Input

The subroutine is called by

```
SELECT #F/xyy
GOSUB' 250 (F, N$, V)
```

where: xyy is the disk device address.
F is the file number.
N$ is the file name.
V is the volume number.

The subroutine displays the prompt MOUNT VOL. XX OF FILE _ _ _ _ _ _ _ _ - UNIT X. After the proper disk is mounted, the catalog index is searched for the file name. If the file name is found, the software header is read to determine if the volume number is correct. A correct volume number causes the subroutine to return control to the application program with the file open.

If the file is scratched, or cannot be found, or the volume number of the file is not the specified volume number, an error message is displayed together with the mount prompt.

Close Input

The subroutine is called by

```
GOSUB' 251(F)
```

where: F is the file number.

The subroutine reads the software trailer and checks whether it specifies an end of file or end of volume. An end of file trailer causes the subroutine to close the file and return control to the application program. An end of volume trailer causes the subroutine to increment the volume counter by one, and initiate the Open Input subroutine with the same file name and the new volume number specified.

CHAPTER 32 DISK SUBROUTINES - MULTISTATION FILE OPEN/END/CLOSE

32.1 OVERVIEW

Multistation File Open/End/Close subroutines, selectable from the Disk Subroutines Menu, are designed for multistation use and multiplexed disk environments. (In previous releases of ISS, Multistation subroutines were called "Multiplexed subroutines".)

Multistation disk files provide additional Security features not available with regular disk files. The Security features are accomplished by using only the Multistation Open/End/Close subroutines on multistation files, instead of regular BASIC-2 Language DATASAVE DC OPEN, END, and CLOSE statements. Special information, namely an access table and password, is maintained in a previously unused portion of the catalog trailer record by the Multistation subroutines. Implementing a unique password for a multistation file will allow only those who know the password to access that file. In addition, the access table allows a station upon opening the file, to have Exclusive (private) access to a multistation file in the Exclusive mode, or file access in one of three nonexclusive (public) modes including the Inquiry, Read Only, and Shared modes. The selected access mode is established each time the file is opened and is discontinued when the file is closed. Closing a file terminates file access, whereas opening begins file access, for each station independent of other stations.

A Set/Release Hog mode (disk drive hog) subroutine is also described in this chapter. All ISS Utilities and subroutines are compatible with multistation files.

32.2 PASSWORD USE

When a file is first created by the Open subroutine, this is called an "open new" operation. Whether or not a password will be required by all stations attempting access to that multistation file (later) is determined by the content of the argument P$ when the Open subroutine is called for the "open new" operation. If P$ contains blanks, a password of blanks is required for all stations to open that file. If P$ contains anything other than blanks, that value of P$ must be provided on any call of the Open subroutine for any user to access that file.

When the "open new" operation has been successfully completed, the password Security feature is operable. Once set by the "open new", the password cannot be changed. A password may be implemented on a subsequent open new command to create a new file if not implemented when the file was previously created on an open new.

32.3 CONVERTING TO MULTISTATION FILES

The conversion from a regular disk data file to a multistation disk data file occurs when that file is opened using the Open subroutine. Thus by accessing the file it is converted to a multistation file automatically; however, for the file to remain a multistation file, the Open, End, Close subroutines always must be used instead of the DATASAVE DC statements.

> CAUTION:
>
> The BASIC-2 statements DATASAVE DC END and MOVE destroy the the access table necessary for multistation files. Instead, use the Multistation End subroutine and Copy/Verify ISS Utility or Copy statement respectively. In general, only use Multistation Open/End/Close subroutines when accessing multistation files.

32.4 OPEN SUBROUTINE FOR MULTISTATION FILES (DEFFN' 217)

The Open subroutine, by virtue of its arguments, allows a multistation file to be opened, the access mode defined, and a password (if any) used. When creating a new file, this is called an "open new." When accessing an existing (old) file, this is called "open old." In addition, when accessing an existing file the access mode may be changed; this is called a "reopen old." The Open subroutine replaces the BASIC-2 Language statement DATASAVE DC OPEN and similarly allocates and reserves file sectors on an open new operation.

Access Modes

The four access modes include Inquiry, Read Only, Shared, and Exclusive. In the Exclusive mode, only the station with Exclusive access may open the file (private access). However, in order for that user to have Exclusive access, no other station can have that file open. The Exclusive access mode is required for an open new.

A file may be opened in the Inquiry mode if that file is not currently open in the Exclusive mode to another station. Conversely, a file open in the Inquiry mode keeps stations requesting the Exclusive mode from opening that file.

A file may be opened in the Read Only mode if that file is not currently open in the Shared or Exclusive mode to another station. Conversely, a file open in the Read Only mode keeps stations requesting the Exclusive or Shared Modes from opening that file.

With the Shared mode requested, that file will be opened successfully if that file is not open in the Read Only or Exclusive modes. Conversely, a file open in the Shared mode keeps stations from opening that file in the Read Only and Exclusive modes.

A file may be opened in the Exclusive mode if no other stations are accessing that file. Once a file is opened in the Exclusive Mode, no other stations can access that file until it is closed.

Disk Hog Mode

With the availability of Exclusive file access, disk Hog mode is usually not necessary. Only in cases where more than one file must be opened with Exclusive access on one disk is the use of disk Hog mode usually necessary. (Exclusive access requires that no other stations have that file open in order for the Open subroutine to be successful.)

> NOTE:
>
> Any use of disk Hog mode by the Multistation Open/End/Close subroutines, or disk Hog mode subroutine DEFFN'215, use the $OPEN and $CLOSE statements, and require device (file) slot #15 in the station's device table. Therefore, device slot #15 is reserved exclusively for Multistation Open/End/Close subroutine use.

Opening Two or More New Files

If two or more multistation files will be opened as new files (open new), after checking that enough disk space exists for all files (use ISS Sort Disk Index Utility), the following procedures are recommended:

1. Set disk Hog mode ON (use a $OPEN statement or GOSUB' 215.)

2. Call Open subroutine with file parameters set for the first new file. Hold disk Hog mode. Exclusive access mode is required for any open new.

3. If successful, call Open subroutine with file parameters set for second new file. Hold disk Hog mode.

4. Repeat step 3 for each new file. However, release disk Hog mode (or use $CLOSE) upon opening the last new file.

5. For each file, perform the following steps:

   a) Perform required processing.

   b) Move current sector address to appropriate file location

   c) Call End subroutine.

   d) Call Close subroutine.

GOSUB' 217 Argument Format

Transfer to the Multistation Open subroutine occurs via the statement:

```
GOSUB' 217(F$,F,C,S,A,P$,A1$,H)
```

where:

F$ indicates the file name, which on an open new command must be unique to that disk. On any open old, it must be identical to the previously assigned file name.

F contains the file (device) number, from 0 to 14.

C contains the station number, from 1-48.

S determines the type of open command. If S is greater than zero, this indicates an "open new" command; also, S equals the number of sectors to be allocated for this file. If S=0 this indicates an "open old" command. If S=-1 this indicates a "reopen old" command during which the access mode is changed.

A determines the access mode for this file. A=1 indicates Inquiry mode. A=2 indicates Read Only mode. A=3 indicates Shared mode. A=4 indicates Exclusive mode. During an open new, A is automatically set equal to 4.

P$ contains a password, if one is required. Password content is determined solely by that file's open new.

A1$ contains the disk address in the XYY form.

H indicates, if H=1, that disk Hog mode will be held following this call. If H=0, Hog mode is released (cancelled) following this call. The Hog mode also may be released using GOSUB' 215, as described on a following page.

> NOTE:
>
> Literal address A1$ must coincide with the address currently selected for the file number.

GOSUB' 217 Return Codes

Q$ contains the return code as summarized below in Table 32-1.

Table 32-1. Multistation Open Return Codes

VALUE Q$	INDICATES	CAUSE AND RECOVERY
blank	successful open	Continue
A	access mode conflict	Retry, or wait and retry. If abnormal delay, check if value of argument A is correct. Check if correct file was accessed (values F$,F,C,A1$). Check if that file was accidentally left open (use File Status Report Utility).
D	open command conflicts with file disposition, e.g., open new issued to existing file, open old issued to file already open to this station, reopen old issued to close file, or file not found.	Check if value of Argument S is correct. Check if correct file was accessed (values F$,F,C,A1$). On open old, check if that file was accidentally left open (use File Status Report Utility).
S	insufficient disk space to complete open new allocation.	Retry after reducing value of S if acceptable, or use different disk by changing A1$ and F. Otherwise use Free Unused Sectors Disk subroutine to create more disk space and retry open new with S as is.
P	password conflict	Check value of P$. Retry with correct file password.

32.5 END SUBROUTINE FOR MULTISTATION FILES (DEFFN' 218)

The End subroutine performs the function of the BASIC-2 statement DATASAVE DC END for multistation files. As with the DATASAVE DC END, upon calling the End subroutine, the current sector address must be at the location required for the end-of-file trailer record. As previously mentioned, use of the DATASAVE DC END on a multistation file results in destruction of the access table necessary for multistation files.

Transfer to the End subroutine is via the statement:

```
GOSUB' 218 (F$,F,A1$,H)
```

where:

F$ is the file name.
F is file (device) number, from 0 to 14.
A1$ is the disk address in the XYY form.
H indicates, if H=1, that disk Hog mode will be held or obtained following this call.
If H=0, Hog mode is released (cancelled) following this call.

There are no return codes.

32.6 CLOSE SUBROUTINE FOR MULTISTATION FILES (DEFFN'219)

Closing a file promptly with multistation operation is usually important because of possible access mode conflicts. Once access to a file is no longer needed, that file should be immediately closed, especially if opened with Exclusive access.

To retain a file for later processing the file may be left open if needed. There is no limit as to how many files can be open for one station because station file status is maintained in the file's catalog trailer record and not by each station. Because the access table resides on nonvolatile storage of disk, the station may be cleared or its power switched OFF, and the file will remain open to that station. This type of action, of course, is not recommended. Files should not accidentally be left open for long periods of time.

Transfer to the Close subroutine occurs via the following statement:

```
GOSUB' 219 (F$,F,C,A1$,H)
```

where:

F$ is the file name.
F is the file (device) number, from 0 to 14.
C is the station number, from 1 to 48.
A1$ is the disk address in the XYY form.
H indicates disk Hog mode release or hold.
If H=1 Hog mode is held following
this call. If H=0 Hog mode is released
(cancelled) following this call.

There are no return codes.

32.7 SET/RELEASE HOG MODE SUBROUTINE (DEFFN' 215)

If a Open/End/Close subroutine is chosen, the Set/Release Hog mode subroutine may be used to switch Hog mode without calling an Open/End/Close subroutine. The $OPEN and $CLOSE statements are used to hog the disk drive specified and may be implemented by the user instead of using DEFFN'215.

Transfer to the disk Hog mode subroutine is via the statement:

```
GOSUB' 215 (A1$,M)
```

where:

A1$ is the disk address in the XYY form.

M is the disk Hog mode indicator. If M=1 Hog
mode is set immediately.
If M=0 Hog mode is released
(cancelled) following this call.

There are no return codes.

CHAPTER 33 - TRANSLATION TABLE SUBROUTINES

The translation table subroutines assign specific sets of hex codes to an array so that it may be used as a translation table with the BASIC-2 statement $TRAN. The subroutines do not actually accomplish the translation; they merely initialize the array. The array is Q9$(). It may be initialized for any of the following translations by means of the indicated GOSUB' subroutine call.

TABLE	SUBROUTINE
EBCDIC TO ASCII	GOSUB'201
ASCII TO EBCDIC	GOSUB'202
2200 TO 1200	GOSUB'203
1200 TO 2200	GOSUB'204

The subroutines load without overlap; they may all be loaded at once.

All the subroutines initialize the same array variable, dimensioned as Q9$(8)32. If more than one table is to be used in an application, either the array variable must be changed by modifying the subroutines, or the application program must execute the appropriate subroutine each time a different translation is to be effected.

When translating 1200 to 2200, all nontranslatable codes are translated into hexadecimal FF.

Assuming that D$ contains data to be translated from ASCII to EBCDIC and the program contains the ASCII to EBCDIC translation table subroutine, the following statement sequence could be used to translate D$:

```
 20 DIM Q9$(8)32
110 GOSUB' 202      :REM INITIALIZE TABLE
120 $TRAN (D$,Q9$()):REM TRANSLATE
130 STOP "D$ TRANSLATED"

9748 DEFFN'202
.
. (translation table subroutine ASCII to EBCDIC)
.
9780                                    ...:RETURN
```

PART V

THE SORT-4 SUBSYSTEM

CHAPTER 34 - SORT-4

34.1 INTRODUCTION

SORT-4 is a subsystem for sorting the records in a disk data file and is loaded from disk by a user-written set-up program. The set-up program provides the parameters for the sort and thereby eliminates the lengthy operator/screen dialog otherwise required for entry of the sort parameters. When sorting is complete, SORT-4 can load a specified application program module. SORT-4, therefore, can be used as a subsystem to an application program.

Although SORT-4 requires little operator attention, it must be run in a partition currently attached to a terminal (a "foreground" partition), because screen displays are included. SORT-4 does not access a global partition and requires at least between 9K to 12K to run, depending on the input file type (see Table 3-1).

SORT-4 offers the following features.

1. The user may specify whether a key sort or a full-record sort is to be performed, or permit SORT-4 to decide. Both the key sort and full-record sort provide sorted output records in exactly the same format as their input record counterparts. In addition, a tag sort may be specified, in which case only pointers to each input record's position on the disk are written into the output (or work) file, and not the actual records themselves.

2. SORT-4 will operate in a multistation or disk multiplexed environment, under ISS conventions.

3. Six input file formats are accepted:

   a) an ordinary cataloged data file,
   b) a BAS-1 data file,
   c) a data file opened and closed with ISS OPEN/CLOSE subroutines,
   d) a KFAM-3 file,
   e) a KFAM-4 file, and
   f) a KFAM-5 or KFAM-7 file.

4. The sort key can contain up to 10 fields. They may be alphanumeric or numeric, but their total length must not exceed 64 bytes, not counting control bytes. Sort order may be specified as ascending or descending for each field, and sort keys may be partial fields, that is, a STR() function of an alphanumeric variable.

5. The following input record formats are supported:

    a) packed arrays, where the array-type blocking is packed for writing on disk, in either DC or BA mode.

    b) Contiguous packed records, where each individual record is packed into a contiguous space within an alphanumeric array, which is written on disk in either DC or BA mode.

    c) Variable length records, packed into an alphanumeric array with either a one-byte length indicator (block size up to 256) or a two-byte length indicator (block size greater than 256). The block may be written in either DC or BA mode. TC (Telecommunication) files are supported by a separate variable length record format.

    d) Individual alphanumeric fields in records written in unpacked format, blocked or unblocked, may contain packed subfields.

    In the above record formats, the field form of $PACK is supported. The internal and delimiter forms of $PACK are not supported. A record may contain either one packed array, or any number of packed fields, but not both. In addition to the formats defined for the field form of $PACK, Wang packed decimal format, signed and unsigned, is supported; exponential as defined in the PACK statement is not supported.

    Any combination of record format and file format is generally permitted, with exceptions noted in Table 34-2 and the text following Table 34-2.

6. With output files written to a disk drive specified in the set-up program as being accessible only to this station (not a multistation disk drive), if a full-record sort is specified, the mounting of the output platter may be deferred until the last pass, at which time the input platter may be removed. With a tag sort, deferred mounting is also allowed if the output file is not written to a multistation disk drive and is not the work file. Deferred mounting permits sorting a full disk platter in a dual platter system.

7. The programmer may write a special input procedure, to be overlaid in Pass 1, to process or screen individual records before input to the sort.

8. SORT-4 treats arrays in input records as arrays. An input record may contain up to 255 fields, each array element counting as one field, provided that the record can be described in not more than 60 table entries (see below).

9. SORT-4 allows a full-record sort on records up to 256 bytes, packed. It also allows a full-record sort with partial fields as sort keys. In most cases, a full-record sort will be faster than a key sort.

10. For KFAM files, a starting and ending key is specified, instead of a starting record number and number of records to be sorted. The input KFAM file is accessed according to key sequence (FINDFIRST/FINDNEXT) instead of sequentially (physical records irrespective of their key values).

In addition to the cataloged input file, SORT-4 requires a cataloged work file. A procedure for calculating the number of sectors required for the work file is described in Section 34.5.8.

SORT-4 may be loaded directly from the appropriate ISS platter at the time of execution. In this case, the ISS platter containing SORT-4 software must remain mounted throughout the sorting procedure. However, it is not recommended that the unused sectors of that ISS platter be used for the work file. Therefore, to maximize available disk space, it is recommended that SORT-4 be copied from the ISS diskette platter it is issued on prior to use. A Copy/Verify reference file has been included to facilitate copying SORT-4 using the ISS Utility Program COPY/VERIFY (indirect input mode). The name of the reference file is ISS.REF5.

Before attempting to sort a file, the programmer should know exactly how that file's records were written to disk. If this information is not available, the ISS Utility Program DISK DUMP should be used to print a portion of the file. The printed output may provide enough clues about the file's contents to enable the programmer to successullly define sort parameters. With records written in DC or DA mode, the DISK DUMP option "Data File Structure" is especially helpful.

SORT-4 Modules

The SORT-4 program modules and functions are as follows:

SORT4: SORT-4 set-up phase, start.

SORT400B: Overlay SORT4 if KFAM input.

SORT400C: Required for multistation files.

SORT401A: Set-up, continued. Process record format and sort key specifications.

SORT402A: Set-up, continued. Calculate length of generated code, available memory, sort blocking and array sizes, and work file size.

SORT402B: Open output file.

SORT403A: Start generating code for modules SORT420A and SORT425A.

SORT404A: Full-record sort or tag sort, finish generating code for module SORT425A.

SORT405A: Key sort only, generate code for modules SORT420A and SORT430A.

SORT406A: Generate code for Pass 1, module SORT410A.

SORT407A: Generate code for Pass 1, module SORT410A.

SORT410A: Pass 1, internal sort.

SORT411A: Overlay SORT410A if KFAM input.

SORT420A: Pass 2, merge, key sort only.

SORT425A: Pass 2, merge, full-record sort or tag sort.

SORT430A: Pass 3, read input file via pointers and write output, key sort only.

SORT490A: Ending (all paths lead to here), close files, stop or load user program to follow.

ISS.REF5: ISS Copy/Verify Reference File

Minimum System Configuration

Station requirements (minimum) include 9K memory for sorting a sequential (non-KFAM) file and between 11K to 12K for KFAM files, depending on the size of the set-up program. A disk, flexible disk, minidiskette, or diskette is also required. SORT-4 programs require about 270 sectors, and thus will not run reasonably on a minidiskette.

34.2 WRITING THE SET-UP MODULE

In order to call the SORT4 program file (module) and provide the necessary sort parameters, the user must write a set-up program as described below. In general, lines 10-179 of the set-up program are executed before SORT4 is loaded, and must be cleared when SORT4 is loaded. Lines 3400-3699 are used to set up the variables for SORT4, and remain after loading SORT4.

The SORT-4 "master" set-up program appears in Table 34-1 and includes each possible statement line required by SORT-4. Some statement lines have default values. If a statement line's default value is acceptable for a particular sort operation, that statement line need not be included in the set-up program. However, if a statement line does not have a default value or if the statement line's default value is not acceptable for a particular sort operation, that statement line must be included in the user's set-up program. Those statement lines whose CONTENTS indicate "KFAM only" need to be included only when the input file format is a KFAM file.

For each statement line in Table 34-1, the entry under the column "DEFAULT VALUE" indicates whether or not a statement line has a default value, by the following conventions:

a. If a statement line has no default value, "none" appears under the "DEFAULT VALUE" column.

b. If a statement line has a default value, either (1) the default value appears under "DEFAULT VALUE" or (2) a hyphen (-) appears under "DEFAULT VALUE" and the default value appears under the "CONTENTS" column.

Be sure to read all sections in the remainder of this chapter before writing the first SORT-4 set-up module.

> **NOTE:**
>
> Any disk device address referred to as "multistation" below indicates a disk drive accessible to any other station besides the station in which SORT-4 is run. If the disk drive is multiplexed, it is a "multistation" disk drive.

Table 34-1. SORT-4 Master Set-up Program

LINE	CONTENTS	DEFAULT VALUE	SEE SECTION
10	REM program identification	none	
20	DIM K(10),B(10),N(10),N$(4)8,P$(4)16, F$(6)3,A$(4)62,M0$(1)21 (This line is necessary to define any arrays referred to below. It may be cleared when "SORT4" is loaded because "SORT4" defines all necessary arrays.)	none	
179	LOAD DC (device) "SORT4" 10, 179	none	
	Machine configuration, see Section 34.5.1 below:		
3400	M = memory size, K bytes. For a 2200MVP specify M = SPACEK + 2. For a 2200VP, specify M = SPACEK. default = 16, 2200VP; or S if under ISS	-	34.5.1
3405	M0$(1) = "table of device addresses" default = "310320330350B10B20B30"	-	34.5.1
3410	SELECT DISK (first device address), #1 (second device address), etc. default = SELECT DISK 310, #1 320, #2 330, #3 350, #4 B10, #5 B20, #6 B30	-	34.5.1
	Sort specifications:		
3415	F = input file format	none	34.3
	0 - unlabeled sequential 1 - BAS-1 labeled sequential 2 - ISS labeled sequential 3 - KFAM-3 4 - KFAM-4 5 - KFAM-5 or KFAM-7		
3420	N$(1) = input file name	none	
3425	F$(1) = input device address (hog mode address if multistation)	none	34.5.2
3430	P$(1) = input file password, if any	blank	34.5.3
3435	J = key file number (KFAM only)	0	
3440	F$(2) = key file device address (KFAM only) (hog mode address if multistation)	blank	34.5.2
3445	B = records per block	1	34.4

Table 34-1. SORT-4 Master Set-up Program

LINE	CONTENTS	DEFAULT VALUE	SEE SECTION
10	REM program identification	none	
20	DIM K(10),B(10),N(10),N$(4)8,P$(4)16, F$(6)3,A$(4)62,MO$(1)21 (This line is necessary to define any arrays referred to below. It may be cleared when "SORT4" is loaded because "SORT4" defines all necessary arrays.)	none	
179	LOAD DC (device) "SORT4" 10, 179	none	
	Machine configuration, see Section 34.5.1 below:		
3400	M = memory size, K bytes. For a 2200MVP specify M = SPACEK + 2. For a 2200VP, specify M = SPACEK. default = 16, 2200VP; or S if under ISS	-	34.5.1
3405	MO$(1) = "table of device addresses" default = "310320330350B10B20B30"	-	34.5.1
3410	SELECT DISK (first device address), #1 (second device address), etc. default = SELECT DISK 310, #1 320, #2 330, #3 350, #4 B10, #5 B20, #6 B30	-	34.5.1
	Sort specifications:		
3415	F = input file format	none	34.3
	0 - unlabeled sequential 1 - BAS-1 labeled sequential 2 - ISS labeled sequential 3 - KFAM-3 4 - KFAM-4 5 - KFAM-5 or KFAM-7		
3420	N$(1) = input file name	none	
3425	F$(1) = input device address (hog mode address if multistation)	none	34.5.2
3430	P$(1) = input file password, if any	blank	34.5.3
3435	J = key file number (KFAM only)	0	
3440	F$(2) = key file device address (KFAM only) (hog mode address if multistation)	blank	34.5.2
3445	B = records per block	1	34.4

Table 34-1. SORT-4 Master Set-up Program (Cont'd)

LINE	CONTENTS	DEFAULT VALUE	SEE SECTION
3565	C$ = "Y" output file cataloged "N" output file not cataloged "W" use work file for output (tag sort only)	Y	34.5.9
3570	P7 = number of sectors in output file	*	34.5.9
3575	P8$ = "K" to force key sort "R" to force full-record sort blank: lets program decide between K and R, or "T" tag sort	blank	34.5.6
3580	D$ = "D" deferred mounting of output disk (full-record sort only)	blank	34.5.9
3585	S2 = station number (required if any disk device is multistation)	0	34.5.1
3590	F$(5) = device address, SORT-4 program modules (hog mode address if multistation)	none	34.5.2
3595	G$ = "name of special input procedure" (blank if none)	blank	34.5.10
3600	M4 = number of bytes occupied by special input procedure	0	34.5.10
3605	M$ = "name of program to load following the sort" (if blank, STOP)	blank	34.5.11
3610	S9 = 0, stop if error, no record count in S8, or 1, stop if error, pass record count in S8, or 2, pass error code in S9 and record count in S8	0	34.5.11
3615	F$(6) = device address for user programs G$ and M$ (hog mode address if multistation)	blank	34.5.2

*Default value described in the specified section.

34.3 INPUT FILE FORMAT REQUIREMENTS

One of the parameters in the set-up program (F) indicates the input file format to be sorted. There are six input file formats referred to as file formats 0, 1, 2, 3, 4 and 5. Sorted output is always written in file format 0.

The terms defined below apply to the following discussion of SORT-4 file formats.

block - Two or more records written to disk as a group to save disk access time. See "blocked" below.

blocked - A file whose records are written in blocks. Contrast with "unblocked."

KFAM - Abbreviation of Key File Access Method. A KFAM file consists of a data file and a "Key File" which indexes records in the data file. KFAM files allow rapid access to records located anywhere in the data file, unlike "sequential files" where record access usually occurs record-by-record from the beginning of the file until the desired record is found.

labeled - Any file containing header information in the first sector, called a "label." Contrast with "unlabeled."

sequential - In this chapter, any non-KFAM file is called a "sequential file."

unblocked - A file whose records are not written in blocks. Typically, one record occupies one sector or more than one sector.

unlabeled - Any file that is not a labeled file.

File Format 0

A cataloged disk file with a hardware END trailer record (DATASAVE DC END) at the end of the live data. File format 0 files are unlabeled, sequential files. If records are blocked, unused spaces in the last block must contain padding records with high or low values in the sort key fields. Records are read sequentially and are included in the sort.

File Format 1

BAS-1 files are labeled sequential files and are known as file format 1 in SORT-4. The file name in the disk catalog is "SCRATCH." The first sector of the file contains a header label. The first field of this label is alpha and contains "HDR", and the second field is alpha and contains the file name, which should match the input file name defined in N$(1). SORT-4 only reads the first two fields of the header label. Remaining fields in the header label may be defined in any way, not necessarily in BAS-1 format.

Data records with file format 1 begin in the second sector of the file. Data must be written in array SORT-4 record format or packed array SORT-4 record format (see Section 34.4). The first field in each record must be a 2-byte alphanumeric field, the "Record ID". The second byte of the Record ID is always "1" for an active data record. An end-of-file condition is recognized when the second byte of the Record ID is greater than "1". (The second byte of the Record ID contains "2" for end-of-file and "3" for end-of-volume. SORT-4 only sorts one volume of a multivolume file and therefore interprets any value greater than "1" as an end-of-file condition.)

The sector following the end of active data contains a special software trailer label. This record is ignored by SORT-4.

Following the software trailer is a hardware END trailer (DATASAVE DC END) which must have been written, because it is used by SORT-4 to determine the file size.

File Format 2

Referred to as "ISS format" because of the label conventions used in the ISS-2 and ISS-3 OPEN and CLOSE Output/Input subroutines (see Sections 31.6 and 31.7), file format 2 is a more general labeled file convention than format 1.

File format 2 is a labeled sequential file, with the first sector a header label, followed by data records, followed by a special software trailer label occupying one sector, followed by a hardware END trailer (DATASAVE DC END).

The header label contains at least two alpha fields. The first field contains "HDR", and the second field contains the file name which must match to the file name in the disk catalog and the input file name defined in N$(1). Remaining fields in the header record are not read by SORT-4, and may contain anything.

The only restriction on the data records is that they must conform to one of the conventions defined in Section 34.4 below.

End-of-file is recognized as follows:

1) If the first field of the record is alphanumeric, a value of HEX(FF) in the first byte of the field signals that the end of the file has been reached.

2) If the first field of the record is numeric, a value of exactly 9E99 in that field signals that the end of the file has been reached.

In the case of packed records, these conventions are interpreted as follows. For contiguous packed records (record formats P, T, and V) HEX(FF) in the first byte of the record, as packed, signals end-of-file. For packed array records (record format A), the record is first unpacked, and then the value of the first field is tested. If the first field is numeric, the packing format must be large enough to accommodate the number 9E99.

The software trailer record is not read by SORT-4. The trailer record is used to indicate end-of-file ("EOF" in the first field) or end-of-volume (some other value in the first field). SORT-4 will only sort one volume at a time, and therefore end-of-file and end-of-volume are both treated as end-of-file.

The hardware trailer (DATASAVE DC END) following the software trailer is required, so that SORT-4 can determine the file size from the "sectors used" entry in the disk catalog.

File Formats 3, 4, 5

KFAM files are denoted by SORT-4 file formats 3, 4, and 5. KFAM-3 files and KFAM-4 files are denoted by SORT-4 file formats 3 and 4 respectively. KFAM-5 and KFAM-7 files are both denoted by SORT-4 file format 5. The KFAM file is read using a special version of FINDFIRST and FINDNEXT.

KFAM-4, KFAM-5, and KFAM-7 files are accessed in the "read only" mode. "Read only" is not defined for KFAM-4, and acts as "exclusive" mode, except where two stations are sorting the same file using SORT-4.

Deleted records are not included in the sort, regardless of whether they are flagged in the User File as deleted.

34.4 INPUT RECORD FORMAT REQUIREMENTS

SORT-4 will sort a variety of packed record formats that previous sort utilities would not handle. Information applicable to combinations of SORT-4 file formats and SORT-4 input record formats appears in Table 34-2.

The following sort parameters combine to determine the exact record format:

```
    B = records per block
   B$ = normal (DC) mode or BA mode
   F$ = record format
   N6 = maximum length of variable portion of record (if applicable)
 A$() = description of fields in record
```

Records Per Block

Records per block (B) must be defined for all record formats except variable length records (F$ = "V" or F$ = "T"). The default value = 1. The maximum value = 255.

DC or BA Mode

Read/write mode (B$) is set to blank for records written in normal DC (or DA) mode, with control bytes separating read fields. Otherwise, set B$ = "B" to indicate records written in BA mode. The default value is blank. Either value of B$ is permitted with any record format F$.

If B$ = blank, SORT-4 reads a sample record or block of records from the file to be sorted and determines the format as written on disk by analyzing the control bytes. The sample record (or block of records) is the first record or block of a sequential file, or the first record or block to be sorted in a KFAM file. The format determined here must be consistent with the parameters supplied by the user.

If B$ = "B", the array to be read in BA mode is always AO$(4)64. Parameters supplied by the user must be consistent with this read field format, e.g., record length not greater than 256 bytes.

Record Format

SORT-4 assumes that all records in the file are written in exactly the same format. Note that packed records of different formats can be written so that they all appear, to the sort, to be written in the same format.

SORT-4 will sort packed records using the field form of $PACK in any of the formats listed below. It can convert any of the numeric $PACK formats, field form, into sort format for sorting. SORT-4 can also convert a signed numeric created by PACK (fixed point only) into sort format for sorting.

SORT-4 record formats are indicated by the value of F$. If F$ = blank (array format), the record is not packed as an array, however, individual fields within the record may be packed. If packing crosses element boundaries, however, F$ must be set not equal to blank. If F$ is not equal to blank ("A", "P", "T", or "V"), the entire record must be packed into a single array, and individual fields may not be packed. The single array must be composed of at least two elements.

In order to discuss the various record formats, it is necessary to differentiate between "read fields" and "packed fields." "Read fields" are the fields as they are written and read from the disk. "Packed fields" are those fields which are obtained by unpacking a read field or array as read from the disk.

A read field which is unpacked into packed fields is defined as a "read only field," which means that once it has been unpacked, it doesn't enter into any further processing. (It may be thought of as a kind of buffer area.)

It should be noted that $UNPACK successfully converts numeric data into a variety of forms, whereas $PACK converts an internal numeric variable into one fixed format only. This means that if numeric data is created by some other means than $PACK, its precise value can change if it is converted via $UNPACK and then $PACK again. Therefore, SORT-4 guarantees to reproduce the identical packed field only if it was originally created with a $PACK statement. Otherwise it will output a field of the same numeric value, if enough space is allowed, but not necessarily the identical format.

Caution is advised in using ASCII free format. In particular, $UNPACK will convert a four-byte field "9E99" to the correct numeric value, but $PACK requires 15 bytes to convert 9E99 back to "9.0000000000E+99".

Caution is also advised in using fields created with the PACK instruction (Wang signed pack format). The PACK instruction works with half bytes, not full bytes. If the PACK image calls for an odd number of half bytes, the last half byte is not used and retains its previous value. If that half byte happens to be set to a hex value A-F, it will cause SORT-4 to stop with ERR X75. The reason for this is that for the sake of consistency in defining fields, the packed field is defined in terms of full bytes, not half bytes, in SORT-4. This field length is converted to an image which will UNPACK the full number of bytes. If the last half byte happens to be HEX A-F, the program will fail. To prevent this from occurring:

a. Create Wang signed packed fields always with an odd number of #'s following the sign.

b. Or clear the alpha field to blanks, to create a trailing zero, before using PACK.

F$ = Blank, Array Format (default value)

"Array format" means that records are written noncontiguously (array blocking) in the normal 2200 mode as required by previous sort utilities, for example:

```
DATASAVE DC#1, A$, B, C$, D
```

indicating four fields per record: alpha, numeric, alpha, numeric.

Blocked records must be written in array form, for example,

```
DIM A$(4)16, B(4), C$(4)30
DATASAVE DC #1, A$(), B(), C$()
```

indicating 4 records per block, each record containing an A$, B, and C$.

The record may also be written in BA mode if the record or block can be expressed as A0$(4)64. This would be either one record per block or four records per block, of 64 bytes each. Warning: 2 records per block in this format would be interpreted by SORT-4 as A0$(1) and A0$(3) in one record, and A0$(2) and A0$(4) in the second record. For contiguous blocking, see record format "P".

Individual alphanumeric read fields in this record format may also contain packed fields. In this case, the complete record must be described in A$() (see below).

If a record in array format contains only read fields, the record description in A$() may be omitted. Partial fields to be used as sort keys may be defined in N() and B().

F$ = "A", Packed Array Format

Records are processed internally in noncontiguous array format, i.e.,

```
DIM A$(4)16, B(4), C$(4)30
```

indicating 4 records per block, with each record containing an A$, B, and C$. But they are packed using the field form of the $PACK statement into one alphanumeric array when written on the disk:

```
$PACK(F=F$) P$() FROM A$(), B(), C$()
DATASAVE DC#1, P$()
```

The record format must be described in A$() (see below) to identify the packed fields in the record.

This format assumes that records are blocked (B = 2 or more). If records are not blocked, use format "P" below.

F$ = "P", Contiguous Packed Records, Fixed Length

A record or block of records is packed into one alphanumeric array. If records are blocked, each record occupies a contiguous space in the array, for example:

```
Record:  DIM A$16, B, C$30
Packing format:  HEX (A0105205A01E)
Record length, packed:  51
Blocking: B = 5
Packed Array:  DIM P$(4)64
```

The first record in the block is packed into bytes 1-51 of the array P$(), the second record is packed into bytes 51-102, etc. The Nth record in the block is unpacked as follows:

```
$UNPACK(F=F$) P$() < (N-1)*51+1, 51 > TO A$,B,C$
```

The record format must be described in A$() (see below) to identify the packed fields in the record.

F$ = "V", Variable Length Packed Records

Variable length records can be sorted if they are written according to the following conventions:

Variable length records must be written in a fixed length block which is read or written as one array. The block may be written in BA mode, in which case the block length is 256.

Depending on the block length, either 1 or 2 bytes are used to indicate the record length and the total bytes written in the block. If the block does not exceed 256 bytes, one byte is used for the length. If the block exceeds 256 bytes, two bytes are used. A separate record format is provided for TC (Telecommunications) variable length records, F$="T" (see below).

The block of records starts with a block length indicator of one or two bytes, indicating the total length of all information written in the block. Each record starts with a record length indicator of one or two bytes (depends on block length), indicating the record length. The record length value includes the length (1 or 2) of the record length indicator. The block length includes the sum of all record lengths plus the length (1 or 2) of the block length indicator, except where the block is written in BA mode (256 bytes), in which case the length of the block length indicator is not counted (maximum 255).

For example, the layout of a block of variable length records would look like this:

Starting Byte	Length	Contents
1	1	Block length indicator (hex), value = 4+R1+R2+R3*
2	1	Record length indicator (hex), value = R1+1
3	R1	First variable length record
3+R1	1	Record length indicator (hex), value = R2+1
4+R1	R2	Second variable length record
4+R1+R2	1	Record length indicator (hex), value = R3+1
5+R1+R2	R3	Third variable length record
5+R1+R2+R3	-	Unused space

*If BA mode, block length = 3+R1+R2+R3

Variable length records must contain a fixed portion followed by a variable portion. The fixed portion is fixed length and fixed format throughout the file and must be described in A$() (see below). The sort key must be contained in the fixed portion of the record. The variable portion always follows the fixed portion and may range from 0 to N6 bytes long, where N6 is the maximum length of the variable portion of the record. N6 must be set for variable length records because its default value is 0. N6 may be set to 0, indicating fixed length records in the variable length record format.

F$ = "T", Telecommunications (TC) Variable Length Packed Format

Although Telecommunications (TC) format may be called a "file format" in general terms, SORT-4 instead treats TC format as a record format, F$ = "T". SORT-4, with some exceptions, allows any combination of record format and file format, but the file format usually specified for TC format is sequential unlabeled files, or F=0. Sequential labeled file format, F=2, is also supported for TC record format, although this combination of record format and file format is rare.

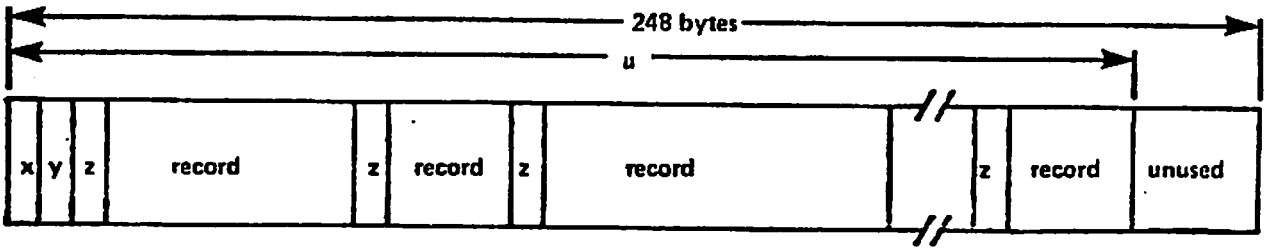
The TC format consists of variable length records contained within a 248 byte block including several control bytes. It is the user's responsibility to ensure that one or several entire variable length records are contained within this block. Individual fields written to disk as variable length records are not sortable as is, and may be reformatted by a user's application program for subsequent SORT-4 input as F$="T" or F$="V" variable length record formats.

The SORT-4 TC record format (F$="T") requires hardware trailer END record (DATASAVE DC END) following the last data sector.

TC format records can be sorted if written according to the conventions adhered to by Wang software for the TC format, which include the following:

TC record format resembles the variable length packed record format, F$="V". Variable length records in TC format are packed into a one dimensional alphanumeric array of four array elements, whose lengths are each 62 bytes, e.g., DIM A$(4)62. The array is saved into a single sector using either DATASAVE DC or DATASAVE DA, and read using DATALOAD DC or DATALOAD DA disk statements.

In packing the records into the array, array element boundaries are ignored; the array is treated as if it were simply 248 contiguous bytes of storage. Within the 248 bytes, three control bytes are used, shown as x, y, and z in the following illustration.



x a one-byte hexadecimal code indicating whether this sector is the last sector, x=HEX(FO), or is not the last sector, x=HEX(00).

y a one-byte hexadecimal value indicating: "the number of used bytes plus one" in the array. In the above illustration where U is the total block length written, y is the hexadecimal equivalent of U + 1. The maximum decimal value of y is 249.

z a one-byte hexadecimal value preceding each record, indicating the record length in bytes.

These variable length records must contain a fixed portion followed by a variable portion, in order to sort. The fixed portion is a fixed length and fixed format throughout the file and must be described in A$( ) (see below). The sort key must be contained within the fixed portion of the record. The variable portion must always follow the fixed portion and may range from 0 (zero) to N6 bytes long.

N6 is the maximum length of the variable portion of the record, and must be set for variable length records. The default value is 0 (zero). N6 may be set to 0, indicating fixed length records in the variable length record format.

For example, the following table represents the layout of TC record format file with three records:

Starting Byte	Length	Contents
1	1	Indicates if this sector is the last sector (hex), HEX(00) = not last sector, HEX(FO) = last sector.
2	1	Block length indicator (hex), value = 6+R1+R2+R3.
3	1	Record length indicator (hex), value = R1 + 1.
4	R1	First variable length record.
4+R1	1	Record length indicator (hex), value = R2+1.
5+R1	R2	Second variable length record.
5+R1+R2	1	Record length indicator (hex), value = R3+1.
6+R1+R2	R3	Third variable length record.
6+R1+R2+R3		Unused space.

Combinations of SORT-4 File Formats and Record Formats

Most combinations of input file formats and input record formats are supported. Table 34-2 (below) provides a cross reference of record/file formats supported by SORT-4.

Table 34-2. SORT-4 Input Record/File Format Combinations

INPUT RECORD FORMATS (F$)	INPUT FILE FORMATS (F)			
	F=0 (general)	F=1 (BAS-1)	F=2 (ISS)	F=3,4,5 (KFAM)
F$="blank" (Noncontiguous unpacked array format)	Supported	Supported	Supported	Corresponds to KFAM record type "A". May correspond to KFAM record types "M" or "N" (see below).
F$="A" (Noncontiguous packed array format)	Supported	Supported	Supported	Supported (rare).
F$="P" (contiguous packed format)	Supported	Not supported	Supported	Corresponds to KFAM-5 and -7 record type "B". May correspond to KFAM record types "C", "M" or "N" (see below).
F$="T" (TC variable length contiguous packed format)	Supported	Not supported	Supported (rare)	Not supported.
F$="V" (Variable length contiguous packed format)	Supported	Not supported	Supported (rare)	Supported (rare). See below.

Comments on Table 34-2

Input record formats and file formats are described in Section 34.3 and Section 34.4. The following notes apply to the combinations of KFAM file formats and SORT-4 input record formats described in Table 34-2.

a. With KFAM files (file format 3, 4, or 5), determination of the record format is related to the "KFAM record type" chosen during INITIALIZE KFAM FILE (a KFAM utility program). Certain KFAM record types apply to certain SORT-4 record formats (F$) as noted in Table 34-2 and below.

   1) KFAM record type "A" always corresponds to F$="blank".

   2) KFAM-5 and KFAM-7 record type "B" always corresponds to F$="P"; B$ must equal "B" in the set-up program.

   3) KFAM record type "C" corresponds to F$="P" if the records contain only alphanumeric fields dimensioned equal in length. KFAM type "C" records which contain any numeric fields or contain all alphanumeric fields that are unequal in length are not sortable.

   4) KFAM record type "M", DC mode, corresponds to F$="blank" if records are unpacked, or F$="P" if records are packed.

   5) KFAM record type "M", BA mode, is not sortable because KFAM type "M" implies more than one sector per record (multiple sector record).

   6) KFAM record type "N" depends on the record's contents. F$ may equal "blank"; or, F$ may equal "P" with B$ equal to either "blank" or "B" in the set-up program.

b. Under certain conditions, KFAM files are sortable as variable length records (although KFAM itself does not support variable length records). That is, F$ may equal "V" if the following conditions are met:

   1) The record length is less than one sector, and records (if blocked) are blocked in a fixed block of exactly one sector.

   2) With blocked records, the third byte of the KFAM pointer (in the Key File) must point to the starting byte of the record in the block. This is the variable Q in KFAM (starting position or "length byte").

Description of Fields in a Record, A$()

The array A$() is provided to describe the record format in detail. Array A$() must be used when F$ = "A", "P", "T", or "V", and also if F$ = blank and it is necessary to define packed fields within a read field. Array A$() is dimensioned as A$(4)62.

Syntax rules for A$() follow:

1. The entire record must be described in A$(), if it is used.

2. Colons or semicolons are used to separate read fields. Commas are used to separate packed fields. (With formats "A", "P", "T", and "V", commas are used exclusively).

3. Fields must be described in the order in which they appear in the record.

4. The record description may not cross element boundaries in A$(). If the description is to be continued from one element of A$() to the next, the first element must end with the correct punctuation mark (colon, semicolon, or comma).

5. Blanks are ignored in A$().

6. Fields are defined as follows:

nnn		Alpha read field
#		Numeric read field
Annn		Alpha packed field
Fnnn		ASCII free format
Innn.dd		ASCII integer format
Dnnn.dd		IBM display format
Unnn.dd		IBM USASCII-8 format
Pnnn.dd		IBM packed decimal format
Snnn.dd		Wang signed packed field, fixed point format only
Wnnn.dd		Wang unsigned packed field, fixed point format only
nnn	=	field length, bytes, in packed form
.dd	=	decimal positions (ignored by SORT-4)

Following any field definition:

(sss) = array of dimension sss

Packed fields can be defined as alpha if they are not used in the sort key, or if their sort sequence would be the same packed or unpacked. Wang unsigned packed fields are always treated as alpha by SORT-4. The fewer packed fields that are defined, the faster SORT-4 will run.

7. Blocking of records is not defined in A$(). Only the individual record should be defined. SORT-4 will construct the necessary arrays based on the description in A$() and the blocking factor, B.

8. For variable length records, only the fixed portion should be defined in A$(). The variable portion is treated by SORT-4 as an alpha field of maximum length N6 (length of N6 is limited only by the size of the block).

9. The maximum number of fields in a record, whether defined by A$() or not, is 255. Each array element counts as one field. The description of contiguous packed records (F$ = "P", "T", or "V") may be abbreviated by combining fields which are not used in the sort. For array-type records (F$ = blank or "A"), 255 fields per record is an absolute limit.

10. The maximum number of entries in A$(), plus one implied read field entry for each group of packed fields defined, is 60. Arrays count as one entry.

11. The maximum lengths of fields of the various formats and the field lengths when converted to sort format, are provided in Table 34-3 below.

Table 34-3. Maximum Field Lengths and SORT-4 Field Lengths

Code	Type	Name	Max Length	Sort Length
00	-	Alpha read	124	L
01	#	Numeric read	8	L
02	A	Alpha packed	124	1
03	F	ASCII free	16	8
04	I	ASCII integer	14	INT (1.5+L/2)
05	D	IBM display	13	INT (2+L/2)
06	U	IBM USASCII-8	13	INT (2+L/2)
07	P	IBM packed	7	1+L
08	S	Wang signed packed	7	1+L
09	W	Wang unsigned packed	7	L*
-	-	Variable length	no limit	not allowed

L = Field length in the above table.
* Wang unsigned packed treated as alpha.

Examples of A$() Syntax

1. Assume array-type blocking, with 4 records per block, where the records are written as follows:

```
DIM A$(4) 16, B(4), C$(4)30
DATASAVE DC#1, A$(), B(), C$()
```

SORT-4 set-up program parameters would be:

```
   B = 4
  F$ = blank
A$(1) = "16;#;30" (not necessary in this case)
```

Note that the default value for B$ (DC mode) is automatically used because it is not specified in the set-up program. The value of N6 should not be set, or should be set equal to 0 (zero).

2. Same record as above, but C$(X) is constructed as follows:

Start	Length	Contents
1	4	Alpha
5	10	Numeric, IBM display, image HEX(320A)
15	5	Numeric, IBM packed decimal, image HEX(5205)
20	5	Numeric, Wang signed packed, PACK(+#####, ####)
25	6	Numeric array, 3 elements, 2 bytes each, IBM packed decimal, image HEX(5002)

SORT-4 set-up program parameters would be:

```
   B = 4
  F$ = blank
A$(1) = "16;#;A4,D10,2,P5.2,S5.4,P2(3)"
```

Because it is not necessary to indicate the decimal positions, A$(1) can be expressed as:

```
A$(1) = "16;#;A4,D10,P5,S5,P2(3)"
```

3. Assume a record is defined as follows:

```
DIM X$(10)16,Y(10),Z(10)
```

where X$(X),Y(X), and Z(X) comprise one record, 10 records per block.

The block of records is packed into one array to save space on disk. Y() and Z() are converted to IBM packed decimal, lengths 6 and 3, respectively. The block is written in BA mode:

```
DIM P$(4)64, F$6
F$ = HEX(A01050065003)
$PACK(F=F$) P$() FROM X$(),Y(),Z()
DATASAVE BA T#1,(L,L) P$()
```

Record definition for SORT-4 would be:

```
     B = 10
    B$ = "B"
    F$ = "A" (packed array)
A$(1) = "A16, P6, P3"
```

4. Records are the same as above, except they are packed individually from scalar fields:

```
FOR X = 1 TO 10
Calculate X$, Y, and Z
$PACK (F=F$) P$() < (X-1)*25+1 >   FROM X$, Y, Z

NEXT X
DATASAVE BA T#1, (L,L) P$()
```

These would be contiguous packed records and would be defined for SORT-4 as follows:

```
     B = 10
    B$ = "B"
    F$ = "P"
A$(1) = "A16, P6, P3"
```

5. Assume variable length records which each contain an account number of 8 bytes, followed by a transaction code of 2 bytes, followed by variable-length information from 1 to 48 bytes in length depending upon the transaction:

```
    F$ = "V"
 A$(1) = "A8,A2"
    N6 = 48
```

The record may be sorted on only the first two fields, account number and transaction code.

6. Fixed-length, 80-byte card images are transmitted to the 2200 and stored in the TC format. To sort these records, define the record format as follows:

```
     F$ = "T"
A$(1) = "A40(2)"
  (N6 = 0,default)
```

34.5 COMMENTS ON WRITING THE SET-UP MODULE

The following explanations of SORT-4 requirements and conventions are referred to in Table 34-1 within the "master" set-up program by section number. File formats and record formats were previously discussed in Sections 34.3 and 34.4.

34.5.1 Machine Configuration

SORT-4 is written to use the memory size contained in the variable S (for ISS) or if S = 0, to use 16K. SORT-4 runs faster if it knows that more memory is available to it. It is set to support devices 310, 320, 330, 350, B10, B20, and B30, but will support any disk device address with the necessary changes.

There are two ways to change the machine configuration. Either change it permanently in the "SORT4" module itself or change it, for any given sort, in the set-up program. If running in a multistation environment, where memory size and available devices change from one station to another, it may be best to define the machine configuration at run time, depending on the station number. If running in a nonmultistation environment, or if all stations are the same size and access not more than a total of 7 disk devices, it is probably better to change the machine configuration in the "SORT4" module.

Memory size is usually indicated using the SPACEK form of the SPACE function. With a 2200MVP partition, specify M = SPACEK + 2; with a 2200VP, specify M = SPACEK. This convention ensures correct memory size use. (SORT-4 automatically accounts for the memory overhead requirements of a 2200VP.) With a 2200VP, memory size is a number from 16 to 64 which represents the memory size of the station in multiples of 1024 bytes.

The table of device addresses, M0$(1)21, allows up to 7 device addresses to be entered into this table. The table must be accompanied by a SELECT statement, selecting the first device in the table as #0 (SELECT DISK), the second as #1, and so forth. File numbers in the device table are linked to devices, rather than files. Hog mode addresses should not be used in the table of device addresses or the SELECT statements.

To set the machine configuration permanently in module "SORT4":

```
CLEAR
LOAD DC (device) "SORT4"
3150 M = memory size, K bytes
3280 M0$(1) = "valid device addresses, maximum 7"
3285 SELECT DISK (first address in table), #1 (second
     address in table), etc.
SCRATCH (device) "SORT4"
SAVE DC (device) ("SORT4") "SORT4"
```

To set the machine configuration for an individual sort, code the set-up module as shown in lines 3400-3410, in Section 34.2.

If running under ISS with an unknown configuration of disk devices, SORT-4 can be set up from ISS start-up common variables:

```
                S2 = Station number (from 1 to 4)
                 S = memory size (With a 2200MVP, use of M = SPACEK + 2 is
                     recommended instead.)
                S$ = system disk (ISS loading address), #0
S$(2) thru S$(9) = other disk devices.
```

Memory size is already set.

Disk devices can be copied to the table as follows:

```
3405 M0$(1) = S$:
     MAT COPY S$() < 4, 18 > TO M0$() <  4, 18 >
```

Note that ISS allows for 9 disk devices, whereas SORT-4 only allows 7.

The SELECT statements must be generated and loaded as a program overlay, as follows:

```
3410    REM INCLUDE DIM D$(4)64, LINE 20, SO THAT THIS
        PROGRAM CAN BE LOADED.  THIS PROCEDURE USES SORT-4
        WORKING VARIABLES.  FIRST STEP - BUILD SELECT
        STATEMENT TO LOAD AT LINE 3410.
3412    D$(1) = HEX(20FF3410A58E)  :REM SELECT DISK
3414    STR(D$(1),7) = S$  :REM SYSTEM DISK (ISS LOADING ADDR)
3416    REM ,#NDDD FOR REMAINING DEVICES
3418    Y = 1
3420    Y = Y + 1
3422    IF Y > 7 THEN 3438
3424    IF S$(Y) = " " THEN 3438
3426    STR(D$(1),6*Y-2) = HEX(2CD7)  :REM, #
3428    CONVERT Y-1 TO STR(D$(1),6*Y), (#)
3430    STR(D$(1),6*Y+1) = S$(Y)
3432    GOTO 3420
3434    REM END SELECT STATEMENTS, WRITE GENERATED CODE
3438    STR(D$(1),6*Y-2) = HEX(0D0000FE)
3440    Z$ = "ISSG0X0A"  :REM WHERE X=station number
3442    CONVERT S2 TO STR(Z$,6,1), (#)
3444    LIMITS T#0, Z$, X, Y, Z
3446    DATASAVE BA T$#0, (X+1, Y) D$()
3448    REM LOAD SELECT STATEMENTS, CLEARING THIS PROCEDURE
3450    LOAD DCT Z$ 3410, 3450
```

34.5.2 Disk Device Addresses and Multistation Operation

To indicate multistation operation or use of a multiplexed disk device, the device address should be written as a hog mode address by adding 8 in hexadecimal arithmetic to the middle digit ("310" becomes "390", "320" becomes "3A0", etc.). Hog mode addresses identify the device addresses for particular files and are only used in array F$(), not in the table of device addresses, M0$(), described in Section 34.5.1.

If two or more files are on the same device, then the device addresses must be consistent, either multistation (hog mode) or not multistation (not hog mode), as the case may be.

Files on a multistation disk (multistation files) are opened and closed using the ISS multistation disk subroutines (described in Chapter 32), as follows:

The input file is opened in the "read only" mode. If the input file is a KFAM-4 file, it is also opened in the "read only" mode under KFAM-4 conventions ("R" placed in the access table in the KDR record). The input file is generally closed when SORT-4 ends, with the exception of a tag sort, where the input file is left open in "read only" mode. The program processing the output of the tag sort should close the input file when it is finished. The KFAM-4 close is always done at the end of SORT-4, whether the input file is left open or is closed.

The input Key File is neither opened nor closed via the ISS subroutines, consistent with the KFAM-5 and KFAM-7 convention of letting the status of the User File also determine the status of the Key File.

The sort work file is opened in "exclusive" mode and is generally closed when SORT-4 ends, with the exception of a tag sort which uses the sort work file as an output file, whereby the sort work file is reopened in "read only" mode when SORT-4 is finished. The program processing the output of the tag sort should close the sort work file when processing is finished.

The output file is opened in "exclusive" mode and closed when SORT-4 is finished.

If SORT-4 ends with an error condition or is terminated by the operator (Special Function Key 31), all multistation files are closed.

If SORT-4 stops with a "hardware" error message, i.e. ERR I96 or ERR I99, the program should be terminated by depressing Special Function Key 31 to insure that all multistation files are closed.

34.5.3 Password Use

Passwords are required to access the input file and the sort work file, if those files have been created with passwords, and are designated as multistation. Otherwise, this parameter can be omitted.

If the output file was previously cataloged (C$ = "Y") and is a multistation file created with a password, then a password for the output file is required. If the output file was not previously cataloged, then any password supplied here becomes the password assigned to the file (see Section 32.4).

Passwords are ignored on files not designated as multistation.

34.5.4 Sort Key Fields

Up to 10 fields may be included in the sort key. The sort key field(s) control sorting and determine output record order. Individual sort key fields may be alpha, numeric, or any of the packed numeric formats listed in Section 34.4. The sort may be ascending or descending on any individual sort key field. The maximum length of the entire sort key, as packed for sorting, is 64 bytes.

Partial fields, which are equivalent to the STR function of an alpha field, may also be defined as sort key fields.

Key field 1 is the highest-order sort key, key field 2 is the next highest, and so on. The key fields are defined as follows: K(1), B(1), N(1), and STR(X9$, 1, 1) define key field 1; K(2), B(2), N(2), and STR(X9$, 2, 1) define key field 2, etc.

The sequence number of the key field is the position of the field in the record containing the key. If A$() is left blank, the sequence number is determined by the position of the record as written on disk. For example, assume records are written as follows, blocked 3:

```
DIM A$(3)6, C$(3)21, S$(3)48, Z$(3)5
DATASAVE DC A$(), C$(), S$(), Z$()
```

Each record contains the following fields, in order:

```
A$(X) = account
C$(X) = customer name
S$(X) = address
Z$(X) = zip code
```

To sort by zip code (Z$) and customer name (C$),

```
   K = 2
K(1) = 4
K(2) = 2
```

This defines the high-order key as the fourth field in the record and the next highest key as the second field in the record.

To use a partial field for a sort key, assume that bytes 47 and 48 of the address (S$) are the state. To sort by state and customer name,

```
   K = 2
K(1) = 3
B(1) = 47
N(1) = 2
K(2) = 2
```

It is not necessary to define N(1) in this case. The default value is the total number of bytes in the field, starting at B(1). Since the default for B(1) is 1, it is not necessary to specify B(1) = 1 if the partial field starts in the first byte of the field. If neither B(1) nor N(1) has a value assigned to it, then all the bytes in the field starting at 1 (the entire field) are included in the sort key.

If the record is described in A$(), the fields specified in A$() determine the sequence number of the sort key. For example, the same record might be described:

```
A$(1) = "6;21;A46,A2;A5"
```

In this case, the address field (S$) is divided into two subfields. The total number of fields defined is 5. To sort by zip code (last field), K(1) = 5, not 4. To sort by state (last 2 bytes of address field), K(1) = 4, not 3. In this case, not a partial field, but a whole field is used and B(1) and N(1) should not be specified.

If arrays are included in the record, each element of the array is counted as one field to determine the sequence number of a key field.

For example:

```
DIM A$8, N(20), C$12
DATASAVE DC A$, N(), C$
```

To sort by C$, K(1) = 22. To sort by the first two numeric fields, K(1) = 2 and K(2) = 3.

The same rules apply to packed records defined in A$(). For example, if the record above is packed into a contiguous 80-byte record:

```
F$ = HEX(A0085003A00C)
$PACK(F = F$) A0$() FROM A$, N(), C$
```

The record to be sorted could be described in any one of the following ways:

```
1)  A$(1) = "A8, P3(20), A12"
2)  A$(1) = "A8, P3(2), A54, A12"
3)  A$(1) = "A8, P3, P3, A54, A12"
```

In any case, to sort on the first two numeric fields, set K(1) = 2 and K(2) = 3. However, to sort on the last field defined in example 1), set K(1) = 22, whereas to sort on the last field defined in examples 2) or 3), set K(1) = 5.

The fewer fields it is necessary to define, the more efficient the sort will be. Also, the sort is more efficient if sort key fields are defined as scalars instead of array elements. Example 3) above produces a more efficient sort, if sorting on the first two numeric fields, but all three examples will work.

The maximum value for the sequence number of a key field, K(1), is 255 which is also the maximum number of fields allowed in a record.

34.5.5 Sorting Partial Files

With particularly large files, it may be necessary to split the file in half in order to sort it. For example, if a sequential file occupies 9000 sectors, one record per sector, the first half of the file would be sorted by specifying D = 1 and L$ = "4500". The second half would be sorted by specifying D = 4501 and L$ = "ALL". The user must then write a merge program to merge the two halves.

"ALL" (L$) means that all records in the sequential file from the specified starting point (D) to the end of the file are sorted.

The parameters D and L$ apply to sequential files only (file formats 0, 1, and 2). These parameters specify blocks of records rather than individual records to accommodate variable length records. The starting point of a particular block of variable length records is easy to find, but the location of a particular record is not known without a search of the file. If records are blocked, D = 2 means that the sort starts with the second block of records, not the second record. Similarly, L$ = "5" means that 5 blocks of records, not 5 records, are sorted.

The default values D = 1 and L$ = "ALL" cause the entire file to be sorted.

For KFAM files (file formats 3, 4, and 5), D and L$ are ignored and need not be specified. Input records are specified by a beginning KFAM key (A$) and an ending KFAM key (E$). A partial KFAM file may be sorted by specifying the starting and ending key values. For example, to split a file into two halves, first sort all records with KFAM keys A - M and then all records with KFAM keys N - Z. The first sort would specify A$ = "A" and E$ = "N". The second sort would specify A$ = "N" and E$ = HEX(FF). Note that the ending key is a limiting value; a record with this key is not included in the sort. The limiting key value of the first group is the same as the starting key value of the next group.

Specifying the beginning and ending keys can also be used to take advantage of the KFAM key to sort a segment of a KFAM file without reading every record in the file.

The default values, A$ = HEX zeros and E$ = HEX F's, cause the entire KFAM file to be sorted.

34.5.6 Type of Sort

Three types of sorts are available in SORT-4: a key sort, a full-record sort, and a tag sort.

The key sort extracts the sort key from the input record, packs it in sort format, and appends to it a 4-byte pointer to the original input record (2-byte sector address and 2-byte pointer to record or starting byte within the block). The "sort record" (record processed by the sort) contains only the sort key and pointer. When all sort records have been sorted, there is a final pass (PASS 3 - OUTPUT) which reads the sort records in sorted sequence, uses the pointer to locate the original input record, reads the input record, and copies it to an output file in sorted sequence.

The key sort is very fast and efficient through the input, sort, and merge phases, but slows down considerably in the last pass, because it must read the entire input file again in random record sequence.

The full-record sort packs the entire input record into a maximum of five "buckets" of 64 bytes each, where the first bucket is the sort key. Certain fields, such as numeric sort key fields, partial sort key fields, and sort key fields which are extracted from an array, are duplicated in the sort record which is then sorted and merged. On the last pass of the merge, sort records are converted back into the original input format and are written in sequential order into the output file.

The full-record sort is generally slower than the key sort during code generation, reformatting input, sorting, and merging, because there are more fields to be defined and moved around. However, the full-record sort compensates for this because it does not read the input file in random sequence in the last pass.

A key sort can be executed for any file, but a full-record sort can only be executed under the following conditions:

1) The input record occupies one sector or less.

2) There is sufficient space in the sort work file.

3) An array in a blocked record, array-type blocking or packed array type, may not exceed 20 elements or require more than 210 bytes for $PACK or $UNPACK.

If no sort type is specified (P8$ = blank), SORT-4 will decide whether to do a key sort or a full-record sort, taking into account the above factors, the proportion of key size to record size, input blocking, and whether deferred mounting of the output file has been specified. This decision does not always determine the fastest way of sorting a particular file. It is worth experimenting to see which type of sort is faster, especially if very large files are to be sorted.

Set P8$ = "K" to specify a key sort, or P8$ = "R" to specify a full-record sort.

The third type of sort, a tag sort (P8$ = "T"), operates like a key sort except that the output of a tag sort is only the pointers to the original input records and not the full records themselves. A user program can then access the input records in sorted order without having to move the input records to a separate output file. This feature eliminates the output pass required in a key sort and reduces the size of the output file considerably. Tag sort output can be used, for example, as a secondary index to a file, by using BA mode access in conjunction with the SORT-4 output pointers to print sorted input file records.

The exact output of the tag sort follows:

1) The file is sequential, with an "END" record following the last block of pointers.

2) Pointers are written in DC mode, 50 per block:

```
DIM A0$(50)4
DATASAVE DC A0$()
```

3) The format of each pointer follows:

   Bytes 1,2: Absolute sector address (hex) of corresponding input record.

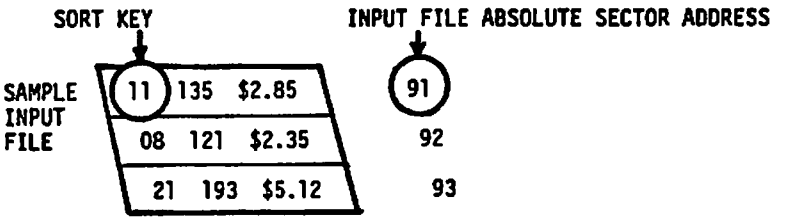
   Bytes 3,4: For array type or packed array type (F$ = blank or "A"), pointer to record within block (hex) in byte 4. For packed records (F$ = "P", "T", or "V"), pointer to starting byte of record within block in bytes 3, 4 (hex). In the case of variable length records, this points to the starting byte of the length indicator.

4) Unused pointers in the last block are padded with HEX(FFFFFFFF).

5) The record count, S8, can be saved to determine the exact number of pointers to be processed (see Section 34.5.11).

Note that if a tag sort is specified, and the input file is multistation, it will remain open in the "read only" mode. The sort work file may be used for the output of a tag sort, eliminating the need for a separate output file (see Section 34.5.9). If the sort work file is multistation, and is used for the output of the tag sort, it will be reopened in "read only" mode, and held open, at the end of the sort.

Figure 34-1 illustrates the basic content and arrangement of three sample input records during each pass of the sort, and for the three types of sorts. Assume the three unblocked input records are located at (absolute) sectors 91, 92, and 93 respectively and are sorted into ascending order of their sort keys. Notice that during passes 1 and 2, the tag sort and key sort are identical. After pass 3, the key sort uses the pass 2 pointers to read the input file records and copy the records in sorted sequence to the output file. The full-record sort, however, carries the entire record for the duration of the sort (in buckets), whereas the key sort carries only pointers (and sort key) through passes 1 and 2. Note that the output of the key sort and full-record sort are identical, and differ greatly from the output of the tag sort.



PASS	TAG SORT	KEY SORT	FULL-RECORD SORT
	4-BYTE POINTER		
1	9100 11 9200 08 9300 21	9100 11 9200 08 9300 21	11 + RECORD IN BUCKETS 08 + RECORD IN BUCKETS 21 + RECORD IN BUCKETS
2	9200 08 9100 11 9300 21	9200 08 9100 11 9300 21	08 + RECORD IN BUCKETS 11 + RECORD IN BUCKETS 21 + RECORD IN BUCKETS
3	9200 9100 9300	08 121 $2.35 11 135 $2.85 21 193 $5.12	08 121 $2.35 11 135 $2.85 21 193 $5.12

Figure 34-1. SORT-4 Sample Operation on Input Records

34.5.7 Construction of Sort Records

In the construction of sort records, there are major differences between SORT-3 and SORT-4 internally. In SORT-3 there may be one or two buckets of up to 64 bytes each, limiting the sort record to 128 bytes. In SORT-4 there may be up to five buckets, allowing up to 256 bytes plus the sort key. In SORT-3, bucket lengths determine how the record is written on disk, and thus affect blocking efficiency in the sort work file. In SORT-4, buckets are packed into array O$() before writing on disk, so that bucket sizes have no effect on sort/merge blocking.

SORT-4 packs the sort record into as few buckets as are required for the particular sort. Therefore it is not necessary to discuss, as in SORT-3, ways of defining dummy sort keys so as to make sorting more efficient. It is all done in the sort program.

The key that is actually used for sorting, by MAT SORT and MAT MERGE, is the entire first bucket. The first bucket starts with the actual sort key, which is followed by nonsort information if it would save a bucket to pack it that way. If it is a key sort or tag sort, the nonsort information is the pointer to the original input record. If the sort key is 60 bytes or less, this pointer is included in the first bucket and acts as a low-order sort key. If a sequential file is being sorted (formats 0, 1, or 2), the pointer keeps records in the original order if duplicate sort keys are encountered.

With a full-record sort, certain fields included in the sort key are duplicated in the nonsort portion of the record:

1) Numeric sort keys.
2) Partial alphanumeric sort keys.
3) Alphanumeric sort keys which are array elements.

This duplication of fields should be taken into account if it is necessary to calculate the length of the sort record. Some of the duplication can be eliminated as follows:

1) Packed numeric sort keys which are always in fixed-point form and always known to be positive can be defined in A$() as alpha.

2) Partial alphanumeric fields used as sort keys should be defined as fields in A$() rather than using the partial field indicators B() and N().

3) Arrays which contain sort keys should be split up in A$() so that the sort key fields are scalars. For example, to sort on the first and fifth elements of K$(20)8, define the record as:

```
A$(1) = "A8, A8(3), A8, A8(15)"
```

This is done automatically by SORT-4 in the case of array-type blocking where A$() is left blank.

34.5.8 The Sort Work File

The sort work file must be cataloged as a disk file prior to running the SORT-4 set-up program. The user may calculate the number of work file sectors required for a particular sort, in order to efficiently allocate disk space for the sort work file. To catalog a disk file, the following can be executed in the immediate mode:

DATASAVE DC OPEN platter, sectors, "name"

To calculate the sort work file size (sectors) necessary for a particular sort, the sort record length must first be known.

With a key sort, the length of the sort record is:

S = K + 4

where S = sort record length.
K = key length (see Table 34-3 for length of numeric keys).

With a full-record sort, the calculation of the sort record length is more difficult. All fields from the input record are packed into the sort record. Numeric sort key fields are repeated in the nonsort portion of the record. Also, alphanumeric sort key fields which are partial fields or elements of an array are repeated in the nonsort portion of the record. The only sort key fields not repeated are alpha scalars where the entire field is included in the sort key.

Packed numeric fields are copied to the nonsort portion of the record in their original alphanumeric form. Internal numeric fields are packed, using the internal form of $PACK.

Variable length records are converted into fixed length records for a full-record sort. Therefore, the length of the sort record includes the maximum length of the variable portion of the record.

The length of the sort record (in bytes) for a full-record sort is:

S = K + I + V - A + N + 3*SGN(N)

where S = sort record length.
K = key length (see Table 34-3 for length of numeric keys).
I = length of input record, or fixed length portion of a variable length record, excluding control bytes. (Numeric field length is 8.)
V = maximum length of variable portion of record.
A = total length of alpha scalar full-field sort keys (not duplicated).
N = number of internal numeric fields in input record.

In the unusual case where there are more than 21 internal numeric fields in a blocked record, 3 additional bytes must be added for every 21 internal numeric fields.

The blocking of sort records varies with the record length and memory space available for arrays, but SORT-4 juggles array sizes to ensure that the blocking is at least 75% efficient. For example, if 248 bytes per sector are available for data, at least 186 bytes will actually be used. Therefore, the space required to store the sort records is:

F = R*S/186

where F = space required for sort records.
R = number of records sorted.
S = sort record length (see above).

Additionally, there is a fixed overhead of 25 sectors for generated code and a variable overhead of one extra block of sort records (up to 16 sectors, in proportion to memory size) plus a String Index which occupies one sector per 36 sorted strings. A total of 50 sectors should accommodate all the overhead, hence the formula:

Sort work file size (sectors) = 50 + F

where F = space required for sort records (see above).

If a tag sort is using the sort work file as an output file, the tag sort output overlays part of the overhead portion of the work file, therefore the sort work file size is the greater of the two:

1) W = 50 + R*S/186
2) W = R/50 + 20 + R*S/186 (tag sort only)

where W = sort work file size, sectors
R = number of records sorted
S = sort record length

If the sort work file is on a multistation disk, it is opened in "exclusive" mode. If it is also used as the output file for a tag sort, it is reopened in "read only" mode at the end of the sort and is not closed.

SORT-4 calculates the work file size required, based upon either the number of records in the input file or the maximum number of records that could be in the input file in the case of variable length records.

SORT-4 will stop if the actual sort work file is not large enough. In certain cases, the actual number of records to be sorted will be much less than the maximum number of records in the file, namely:

1) When a partial file is being sorted using a starting and/or ending KFAM key.

2) When certain records are selected for sorting via a special input procedure (see Section 34.5.10).

3) With variable length records, where the average number of records in a block is less than the number of minimum length records (variable portion zero) that will fit in a block.

The variable P8 is provided to indicate approximately the maximum number of records to be sorted, if that number (P8) is significantly less than the maximum number of records in the file. If P8=0 (default value), SORT-4 uses the maximum number of records in the file to calculate the required sort work file size. If P8 is greater than zero, SORT-4 calculates sort work file size on the basis of P8 records.

In addition to checking the work file size in the set-up phase, SORT-4 also checks each time a block of sort records is written in Pass 1 to make sure that the sort work file space is not exceeded. Therefore if P8 is too small, no damage is done.

34.5.9 The Output File and Deferred Mounting

The output of SORT-4 is always an unlabeled sequential file (file format 0). Output records are written in the same format as input records, and the blocking is the same, with the exception of a tag sort (see Section 34.5.6 for the output format of a tag sort).

The output file may either be previously cataloged by the user (C$="Y") or cataloged by SORT-4 (C$="N"). If a tag sort is specified, the sort work file may be used for output (C$="W").

If the output is on a multistation disk, it is opened in "Exclusive" mode. In a tag sort, if the output file is also the sort work file, it is reopened in "read only" mode and kept open at the end of the sort. Otherwise the output file is closed when SORT-4 is finished.

In a full-record sort or a key sort, the default length of the output file in SORT-4 is equal to the length of the input file.

In a tag sort, the length of the output file is calculated as INT(P8/50) +3, where P8 = the maximum number of records to be sorted (see Section 34.5.8).

The variable P7 is used for the length of the output file, in sectors. If P7=0, the default calculations above are used. If P7 is set to some number greater than zero, then that number is used.

If the output file was not previously cataloged, then exactly P7 sectors are cataloged. If the output file is already cataloged, then it must contain at least P7 sectors.

Under certain conditions, the disk containing the file may be dismounted at the end of Pass 1 and replaced by the disk containing, or to contain, the output file. This permits a file occupying a full disk platter to be sorted using two disk platters. This procedure, referred to as "deferred mounting", is indicated by D$="D" in the set-up module.

Deferred mounting is allowed under the following conditions:

1) The disk containing the output file is not multistation.

2) A full-record sort or a tag sort (not a key sort) is being performed.

3) If it is a tag sort, the output file is not the sort work file.

Normally the input file, output file, sort work file, and the SORT-4 program modules must remain mounted throughout the sort. With deferred mounting, the SORT-4 modules and the sort work file can occupy the fixed disk, while the removable disk is switched from input to output.

If output records are blocked, SORT-4 generates padding records to fill the unused record positions in the last block. SORT-3 generates high values for ascending keys and low values for descending keys, whereas SORT-4 only generates high values.

The padding procedure for the various record formats are as follows:

F$ = blank (array-type blocking) - The first read field and all read fields containing sort keys are filled with HEX(FF) in all bytes if the field is alpha or exactly 9E99 if the field is numeric.

F$ = "A" (packed array) - The first field and all fields containing sort keys are filled with high values before the record is packed. Alpha fields are filled with HEX(FF) in all bytes. Numeric fields are filled with the highest value that can be packed. For ASCII free format, with a field length of 15 or more, the value is 9E99. Other $PACK formats are padded with the number of 9's indicated below, where L = packed field length.

Field Format	Number of 9's
ASCII free	L-1
ASCII integer	L-1
IBM display	L
IBM USASCII-8	L
IBM packed	2*L-1

Wang packed formats are padded as follows:

Signed: HEX(099999...) per field length
Unsigned: HEX(999999...) per field length

F$ = "P", fixed length packed records: The entire record, in packed form, is filled with HEX F's.

F$ = "V", variable length packed records or F$ = "T" TC record format: No padding is necessary.

34.5.10 Special Input Procedure

The user may specify a special input procedure to be overlaid in Pass 1 of the sort (module SORT410A). This procedure can be used to sort records selectively or for any other input processing that does not interfere with the functioning of the sort. No other files may be opened during the input processing phase wherein input records are selected for sorting based on certain user-defined logical relationships between fields.

Coding of the input procedure must conform to the following rules:

The first line of the special input procedure must be 1000. Lines 1000-1999 may be used for REM statements, DIM statements, an initialization procedure, or any other processing to take place before any input records have been read.

Lines 2500-3499 are used for normal input processing. At this point, the input record has been read, but has not yet been included in the sort. To include the record in the sort, GOTO 3500 (or drop through). To exclude the record from the sort, GOTO 2400. Input record selection is graphically shown in Figure 34-2.

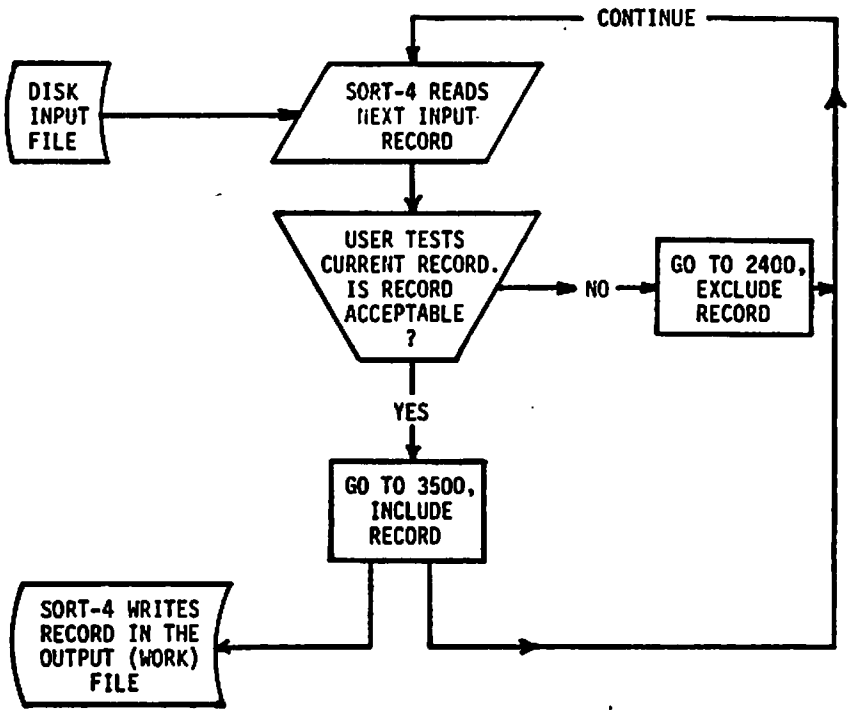


Figure 34-2. Input Record Selection Flowchart

To reference the input record, the variable names assigned by SORT-4 must be known. Perhaps the easiest way to do this is to set up the particular sort with a special input procedure:

```
1000  REM
2500  STOP
```

When the program stops,

```
SELECT LIST 215
LIST 10, 3550
```

Depress Special Function Key 31 to end program, if multistation.

The input record is defined on the following lines:

```
 500  DIM statements
2440  Read input record, $UNPACK if F$="A"
2460  MAT COPY variable length record to fixed work area, O$()
3510  $UNPACK if F$=blank, "P","T", or "V"
```

If records are blocked, the variable Q points to: (1) the record within the block if F$=blank or "A", (2) the starting byte of the record if F$="P", or (3) the length byte(s) preceding the record if F$ = "T" or F$ = "V".

Only packed array records (F$="A") are unpacked prior to the special input procedure. Other formats are unpacked later, so that records not in the proper packed format can be dropped in the special input procedure. Records are available, in the special input procedure, in the following forms:

F$ = blank:	Read fields available, Q points to record within block.
F$ = "A":	Packed fields available, Q points to record within block.
F$ = "P":	Record is available in packed form in array A0$(), starting byte Q.
F$ = "V": or F$ = "T"	Record is available in packed form in array A0$(). Q points to the length byte(s). If full-record sort, the record, starting with length byte(s), has been moved to array O$(), starting byte 1.

Variable names G0-G9, H0-H9, ... L0-L9 may be used as working variables in the special input procedure. (These are reserved for the output record definition and are not used in Pass 1.)

SORT-4 assigns variable names A0-A9, B0-B9, ... F0-F9, to the input record, based upon a table constructed in the set-up phase. If the record has been described in A$(), this table is simply one entry per A$() entry, plus an implied read only field inserted in front of each group of packed fields. Variable names are calculated directly from the table subscript. The name A0 is always assigned to the first entry in the table, A9 to the tenth, B0 to the eleventh, etc. For example, if the set-up program defines

```
A$(1) = "#; A5, P3(5), A4; #(16)"
```

the table entries and variable names are:

Table Entry	Field Type	Field Length	Array Dimension	Variable Name
1	numeric read	8	1	A0
2	read only	24	1	A1$24
3	alpha packed	5	1	A2$5
4	IBM packed	3	5	A3(5)
5	alpha packed	4	1	A4$4
6	numeric read	8	16	A5(16)

The read fields A0, A1$, and A5() are available during the special input procedure. The packed fields may not even be defined by SORT-4, in the case of a key sort or tag sort.

If a numeric read field is designated as a sort key, it is always defined as an array by SORT-4. In the example above, if the first field is a sort key, it is defined as A0(1), not A0. If the field is already defined as an array (array, or blocked, or blocked array), then its definition is the same, whether it is a sort key or not.

In packed array format (F$="A"), packed numeric fields designated as sort keys are always defined as arrays. What would otherwise be a scalar is defined as an array of dimension 1.

If records are blocked as arrays (F$=blank or "A"), SORT-4 adds another dimension for blocking to the variables to which the blocking applies. With array-type blocking, the extra dimension is added to read fields. With packed array format, the extra dimension is added to packed fields. In the example above, where records were written 5 per block, the variable names would become A0(5), A1$(5)24, A2$5, A3(5), A4$4, A5(16,5). The read fields for the current record being processed would be A0(Q), A1$(Q), and A5(X,Q), where X=1 to 16.

For packed records, formats "A", "P", "T", and "V", the read field is always A0$(). The fields defined in A$() are then assigned variable names A1 and up.

In the case of a packed array, all fields are always unpacked and available in the special input procedure. Blocking applies to packed fields. For example:

```
   F$ = "A"
    B = 5
A$(1) = "A12, P3(5), S4, A1(3), F15"
   B$ = "B"
```

Table Entry	Field Type	Field Length	Array Dimension	Variable Name
1	read field	64	4	A0$(4)64
2	alpha packed	12	1	A1$(5)12
3	IBM packed	3	5	A2(5,5)
4	Wang packed	4	1	A3$(5)4
5	alpha packed	1	3	A4$(3,5)1
6	ASCII free	15	1	A5(5)

The fields of the input record available in the special input procedure are A1$(Q), A2(X,Q), A3$(Q), A4$(Y,Q), and A5(Q), where X=1 to 5 and Y=1 to 3.

Note that the Wang packed decimal field (signed or unsigned) is unpacked as an alpha field at this point.

If a record description is not provided in A$() (array-type only), SORT-4 constructs a table based on the format of the record as written on disk. Consecutive fields of the same length and type are combined into arrays, except that fields containing sort keys are always defined as scalars. For example, if the record as originally written was:

```
DIM A$(2)8, B$(2)8, C$(2)8, D$(2)8, E$(2)12
DATASAVE DC A$(), B$(), C$(), D$(), E$()
```

and there are 2 records per block, and the first field is the sort key, SORT-4 will define the record as follows:

```
DIM A0$(2)8, A1$(3,2)8, A2$(2)12
DATALOAD DC A0$(), A1$(), A2$()
```

Individual records may be referenced in the special input procedure as A0$(Q), A1$(X,Q), and A2$(Q), where X=1 to 3.

The variable M4 should contain a conservative estimate of the number of bytes occupied by the special input procedure and any common variables which may be carried through the sort.

Appendix B shows the variables used by SORT-4. The special input procedure should avoid using any of these variables except the ones reserved for output. If common variables are to be carried through the sort, they should be variables not used by SORT-4.

34.5.11 Exit From SORT-4

If M$=blank (default), SORT-4 will stop when finished, displaying a count of the number of records sorted and "END OF SORT", or the appropriate error message.

Or M$ may contain the name of the program to be loaded following the sort. This program must be present on the device specified in F$(6).

If a program is to be loaded following the sort, the following options are available:

S9 = 0 (default), stop if SORT-4 ended in an error condition. Otherwise clear all common variables starting at S8 and load user program M$.

S9 = 1, stop if SORT-4 ended in an error condition. Otherwise store record count in S8 (COM), clear all common variables starting at S9, and load user program M$.

S9 = 2, store error code (see Section 34.7 below) in S9, store count of records sorted in S8, clear common variables starting at M$, and load user program M$.

Common variables are defined in SORT-4 starting with S8, S9, M$8, etc. The value of S9 in the set-up module indicates how many of these common variables will be saved (0, 1, or 2), with the appropriate information, when loading user program M$.

34.6 NORMAL OPERATING PROCEDURE

Except for deferred mounting of the output file, there is no operator dialogue in SORT-4. The input file, sort work file, all SORT-4 modules, and any necessary user modules (set-up program, special input procedure, and program to be loaded following the sort) must be present and cataloged on disks which are mounted prior to the running of SORT-4. The output file may or may not be cataloged, as specified in the set-up module, and the disk which will contain the output may or may not be mounted at the start of SORT-4.

If deferred mounting of the output is specified, the following dialog takes place at the end of Pass 1:

1. REMOVE INPUT VOLUME AND MOUNT OUTPUT VOLUME

   ENTER 'GO' TO RESUME

   Replace the input volume with the output volume. Enter the letters "GO" and touch the RETURN Key when ready.

   If the response is not correct, the prompt reappears, requesting reentry. If the response is correct, the program will display "PASS 2 -- MERGE", and continue with the sort.

During the running of SORT-4, the following information will be displayed on the screen, starting at display screen line 4:

```
(Phase of the sort:  Start, Pass 1, Pass 2, or Pass 3)
INPUT FILE (name) DEVICE (address) FORMAT (number)
RECORDS PER BLOCK (number)
STARTING BLOCK # TO BE SORTED (number) or STARTING KEY TO BE SORTED
    (key), if KFAM
```

```
NUMBER OF BLOCKS TO BE SORTED (number or "ALL") or ENDING KEY (NOT
    SORTED) (key), if KFAM
WORK FILE (name) DEVICE (address)
NUMBER OF KEY FIELDS (number)
KEY FIELDS (field number, D if descending, repeated)
OUTPUT FILE (name) DEVICE (address) CATALOGED (code)
(Type of sort:  KEY SORT, TAG SORT, or FULL-RECORD SORT)
```

If it is necessary at any time to terminate the sort before it is finished, key HALT/STEP and then depress Special Function Key 31 to make sure that all files are closed properly. The program will stop, normally, with the message "OPERATOR INTERVENTION". If error messages are passed to a user program following the sort, it should stop with an appropriate message in that program. SORT-4 error messages appear in Section 34.7, below.

If error messages are not passed to the user program, any error message encountered will be displayed, as listed in Table 34-4. Otherwise, upon successful completion, the following is displayed:

```
RECORD COUNT NNNNN
STOP END OF SORT
```

34.7 ERROR MESSAGES AND RECOVERY PROCEDURES

There are two types of error conditions that could be encountered during the operation of SORT-4. Hardware (ERR Inn) errors, if encountered, are described below under "Hardware Errors".

SORT-4 supplies software-generated error messages which enable the error's cause to be quickly isolated and corrected by the programmer. Software error messages, which appear as several words indicating the nature of the error, are described under "SORT-4 Software Errors" below.

Hardware Errors

Certain "hardware" error messages will cause SORT-4 to stop. If any of these occur, depress Special Function Key 31 to end the sort and close any files that may be open in a multistation environment.

```
ERR I96 Disk read error
ERR I99 Disk read-after-write error
```

If either of the above occur, depress Special Function Key 31 to end the program. Try rerunning the sort. ERR I96 means that the information written on the disk is bad. ERR I99 means that the disk platter itself is bad. If the error recurs, try running from a backup copy of the disk. These errors should occur very rarely if the hardware is within its temperature and humidity environment limits.

ERR D83 The output file to be cataloged is already cataloged. Depress Special Function Key 31 to end the program. Make sure that the correct disks are mounted. This may require a programming change in the set-up module (C$ = "Y").

ERR D82 Indicates "file not found." A data file or program module does not exist on the specified device. Depress Special Function Key 31 to end the program. Make sure that the correct disks are mounted. This may require a programming change in the set-up module.

Other "hardware" errors: This could be caused by hardware or software failure, invalid data, or a number of reasons. Make a note of the line number of the program that caused the error, and the error number. Enter LIST S and make a note of the module name displayed on the first line. Depress Special Function Key 31 to end the program and close all files.

SORT-4 Software Errors

SORT-4 checks for many error conditions and comes to an orderly halt, closing all files, if SORT-4 detects an error. These error conditions are listed below. The number is the error number passed (S9) to a user program, if that option is specified; otherwise the accompanying message is displayed on the screen and SORT-4 stops. (Also see Section 34.5.11.)

Alphabetic Software Error Message List

Error messages displayed by SORT-4 are listed according to alphabetic order below in Table 34-4, with a cross-reference to the number associated with each error message. Error messages and recovery procedures are listed in numeric order In Table 34-5.

Table 34-4. Alphabetic List of SORT-4 Error Messages

ERROR MESSAGE	NUMBER BELOW
BLOCK SIZE TOO SMALL	see #25
DEFERRED MOUNTING INVALID	see #20
DEVICE CONFLICT	see #9
ERROR CLOSING FILES	see #35
ERROR OPENING OUTPUT FILE	see #17
ERROR OPENING WORK FILE	see #15
FULL RECORD SORT NOT POSSIBLE	see #32
INPUT BLOCKING INVALID	see #8
INPUT FILE BUSY	see #39
INPUT FILE OPEN ERROR	see #3
INVALID DEVICE ADDRESS	see #10
INVALID END OF FILE	see #1
INVALID FORMAT	see #2
INVALID NUMBER OF BLOCKS	see #6
INVALID NUMBER OF KEY FIELDS	see #26
INVALID OUTPUT TYPE	see #30
INVALID RECORD DEFINITION	see #23
INVALID RECORD FORMAT	see #13
INVALID RECORD TYPE	see #24
INVALID RECORDS PER BLOCK	see #4
INVALID SORT KEY SPECIFICATIONS	see #21
INVALID SORT TYPE	see #31
INVALID STARTING BLOCK	see #5
MEMORY SPACE TOO SMALL	see #33
NO CPU NUMBER	see #29
NO RECORDS TO SORT	see #11
OPERATOR INTERRUPT	see #28
OUTPUT FILE TOO SMALL	see #19
PACKED ARRAY MUST BE BLOCKED	see #37
PACKED RECORD MUST BE ARRAY	see #38
PROGRAM ERROR	see #34
RECORD COUNT INPUT = XXXXX, OUTPUT = XXXXX, ERROR	see #36
RECORD COUNT = XXXXX; STOP END OF SORT	see #99
RECORD DEFINITION INCONSISTENT	see #22
SEQUENCE ERROR	see #18
SORT KEY TOO LONG	see #27
STARTING BLOCK TOO HIGH	see #7
TOO MANY FIELDS	see #14
WORK FILE BUSY	see #40
WORK FILE TOO SMALL	see #16
WRONG INPUT FILE	see #12

Table 34-5. Numeric List of SORT-4 Error Messages and Recovery Procedures

1. INVALID END OF FILE

   Input file formats 0, 1, or 2 must be ended with a hardware END trailer (DATASAVE DC END). The number of sectors used is invalid.

   Recovery: Correct the input data file.

2. INVALID FORMAT

   Format (F) not 0 - 5 (file format).

   Recovery: Correct the format code (F) in the set-up program.

3. INPUT FILE OPEN ERROR

   Multistation OPEN error (file not found, invalid password, etc.) or KFAM key file not found.

   Recovery: Correct the input data file and/or set-up program (file name, password).

4. INVALID RECORDS PER BLOCK

   Blocking (B) is not an integer from 1 to 255, or doesn't match actual blocking in sample record, or doesn't match blocking (V8$) in KFAM KDR record.

   Recovery: Correct set-up program.

5. INVALID STARTING BLOCK

   Starting block # to be sorted (D) is less than 1 or not an integer.

   Recovery: Correct set-up program.

6. INVALID NUMBER OF BLOCKS

   Number of blocks to be sorted (L$) is not "ALL" and not numeric, or not an integer greater than 0.

   Recovery: Correct set-up program.

7. STARTING BLOCK TOO HIGH

   Starting block # to be sorted (D) is greater than the number of blocks of records in the input file.

   Recovery: Either there are no records to sort, or the set-up program should be corrected.

Table 34-5. Numeric List of SORT-4 Error Messages and Recovery Procedures (Cont'd)

8. INPUT BLOCKING INVALID

The block length, in sectors, does not divide evenly into the length of the data portion of the input file.

Recovery: Correct the input data. Blocks of records must be fixed-length and must be written on disk in the same identical format.

9. DEVICE CONFLICT

A device address is specified as multistation for one file and not multistation for another file. The device address must be consistent (multistation or not multistation) for all files on that device.

Recovery: Correct the set-up program.

10. INVALID DEVICE ADDRESS

A device address is specified which is either blank or not in the table of device addresses, M0$().

Recovery: Correct the set-up program.

11. NO RECORDS TO SORT

Either KFAM returns an end-of-file condition trying to find the first record to sort, or the count of records being sorted, at the end of Pass 1, is zero.

Recovery: None.

12. WRONG INPUT FILE

The file name in the header label, formats 1 and 2, does not match the input file name specified in the set-up program.

Recovery: Mount correct disk and rerun, or correct the set-up program.

13. INVALID RECORD FORMAT

The sample record being used to determine the input record format does not have correct control bytes.

Recovery: Correct the input data file.

Table 34-5. Numeric List of SORT-4 Error Messages and Recovery Procedures (Cont'd)

14. TOO MANY FIELDS

More than 255 fields are defined in the input record, or more than 60 table entries are required to describe it, or more than 256 bytes are required for a DATALOAD or DATASAVE statement to read the input or write the output.

Recovery: Change the set-up module to describe the input record as fewer fields, combining scalars into arrays wherever possible. If this is not possible, SORT-4 will not sort this file.

15. ERROR OPENING WORK FILE

The multistation OPEN subroutine detects an error condition (file not found, invalid password, etc.) trying to open the sort work file.

Recovery: Check that the correct disk is mounted, or correct the set-up program (file name, password, device).

16. WORK FILE TOO SMALL

The work file contains less than 25 sectors, or is too small to sort the maximum number of records specified (P8), or is too small to be used as both a work file and output file for a tag sort, or the dynamic check in Pass 1 shows that the work file is full.

Recovery: Adjust the maximum number of records (P8) in the set-up module, or create a larger sort work file, or switch from a full-record sort to a key sort (if P8$ = "R").

17. ERROR OPENING OUTPUT FILE

Multistation OPEN error (file not found, invalid password, output file presently in use, etc.) trying to open the output file.

Recovery: Check that the correct disk is mounted and the output file is not being accessed by another station. Correct the set-up program (file name, password) if necessary.

18. SEQUENCE ERROR

The sorted keys are not in proper sequence. This is a check against possible hardware or software malfunction.

Recovery: Rerun the program. Notify Wang Laboratories if the error persists.

Table 34-5. Numeric List of SORT-4 Error Messages and Recovery Procedures (Cont'd)

19. OUTPUT FILE TOO SMALL

The output file is smaller than the number of sectors specified in P7 (default = input file size), or the dynamic check in pass 2 or 3 shows that the output file is full.

Recovery: Adjust the output file size (P7) in the set-up module, or create a larger output file.

20. DEFERRED MOUNTING INVALID

Deferred mounting of the output file (D$ = "D") may not be specified if a key sort is being performed, or if the work file is also used for the output of a tag sort, or, for all sort types, if the output file is multistation.

Recovery: Correct the set-up program.

21. INVALID SORT KEY SPECIFICATIONS

The description of sort key fields in K(), B(), N(), and X9$ is invalid, or the description of a partial sort key field is inconsistent with the field length, or a partial sort key field has been specified for a numeric field.

Recovery: Correct the set-up program.

22. RECORD DEFINITION INCONSISTENT

The record definition supplied in A$() is inconsistent with other record definition information. With packed records (F$ = "A", "P", "T", or "V"), A$() is blank, or the sample record shows more than one read field array, or the array is numeric. Also possibly caused by the definition in A$() not fitting the sample record: Too many or too few fields are defined, field lengths of packed fields don't add up to field lengths of read fields, or field lengths, types, and array dimensions don't match the sample record.

Recovery: Correct the set-up program.

23. INVALID RECORD DEFINITION

The record definition supplied in A$() is invalid within itself. The field type is invalid, length or array dimensions out of bounds, punctuation marks are invalid, or there is an invalid sequence of read fields and packed fields. Or, for variable length records, the length of the variable portion (N6) is not an integer or is less than zero.

Recovery: Correct the set-up program.

Table 34-5. Numeric List of SORT-4 Error Messages and Recovery Procedures (Cont'd)

24. INVALID RECORD TYPE (F$)

The record format (F$) is not blank, "A", "P", "T", or "V".

Recovery: Correct the set-up program.

25. BLOCK SIZE TOO SMALL

For fixed-length packed records (F$ = "P"), the product of blocking times record length is greater than the block length. For variable length records (F$ = "T" or "V"), the block is too small to hold the largest possible record.

Recovery: Correct the set-up program.

26. INVALID NUMBER OF KEY FIELDS

The number of sort key fields (K) is not an integer from 1 to 10.

Recovery: Correct the set-up program.

27. SORT KEY TOO LONG

The total length of the sort key exceeds 64 bytes.

Recovery: Change the set-up program to shorten the sort key, if possible.

28. OPERATOR INTERRUPT

SORT-4 was terminated by depressing Special Function Key 31.

Recovery: Rerun the program.

29. NO CPU NUMBER

No CPU (station) number (S2) or an invalid station number was specified with multistation files.

Recovery: Specify the station number (S2) in the set-up program.

30. INVALID OUTPUT TYPE

The work file may not be used as the output file (C$ = "W") except with a tag sort (P8$ = "T").

Recovery: Correct the set-up program.

Table 34-5. Numeric List of SORT-4 Error Messages and Recovery Procedures (Cont'd)

31. INVALID SORT TYPE

The type of sort specified (P8$) is not blank, "K", "R", or "T".

Recovery: Correct the set-up program.

32. FULL RECORD SORT NOT POSSIBLE

A full-record sort has been specified (P8$ = "R" or D$ = "D" forcing full-record sort), but it is not possible to perform a full-record sort for one of the following reasons:

a. The nonsort portion of the record exceeds 256 bytes.

b. A 2-dimensional array, created by an array in a blocked record, is too large to be packed in one $PACK statement.

Recovery: Change the set-up program to perform a key sort or tag sort.

33. MEMORY SPACE TOO SMALL

The memory size specified (M or S) is less than 7K, or the memory size is too small for the following minimum requirements for the sort/merge file:

a. Sort blocking must be at least 5 records per block and occupy at least 2 sectors.

b. Sort blocking must be at least 75% efficient (at least 186 bytes of data per sector in the sort work file).

Recovery: Change the set-up program to perform a key sort or tag sort, or correct the memory size specified. It may not be possible to sort unusually large records or blocks of records in an 8K machine.

34. PROGRAM ERROR

Hardware or software error. Generated $PACK statement does not match generated hex images.

Recovery: Notify Wang Laboratories.

35. ERROR CLOSING FILES

An error is detected by one of the multistation subroutines when writing an END record on the output file or reopening the work file used for output, or closing a file.

Recovery: This should not occur. Rerun the program. Notify Wang Laboratories if this error recurs.

Table 34-5. Numeric List of SORT-4 Error Messages and Recovery Procedures (Cont'd)

36. RECORD COUNT INPUT = XXXXX, OUTPUT = XXXXX, ERROR

The number of input records entered into the sort does not match the number of output records from the sort. Hardware or software error.

Recovery: Rerun the program. Notify Wang Laboratories if this error recurs.

37. PACKED ARRAY MUST BE BLOCKED

If the input record format is specified as packed array (F$ = "A"), then the blocking factor (B) must be greater than 1.

Recovery: Change the set-up program. If packed records are not blocked, specify F$ = "P".

38. PACKED RECORD MUST BE ARRAY

For the packed record formats (F$ = "A", "P", "T", or "V"), the record or block written on disk must be an array containing 2 or more elements.

Recovery: Change F$ to blank in the set-up program, or change the format of the input record on disk so that at least 2 elements of an array are written per block of packed records.

39. INPUT FILE BUSY

Multistation input file cannot be opened in "read only" mode because of an access conflict, or KFAM-4 access conflict or access table full.

Recovery: Rerun the program when the access conflict is resolved. It may be necessary to clear spurious information out of the access table. If no other station is accessing the file, see the ISS Utility FILE STATUS REPORT if a nonKFAM file was used as input, or see the KFAM Utility RESET ACCESS TABLE (either KFAM-3, 4, 5, or 7 version), if a KFAM input file was specified.

40. WORK FILE BUSY

Multistation work file cannot be opened in "exclusive" mode because of an access conflict.

Recovery: Rerun the program when the access conflict is resolved. It may be necessary to clear spurious information out of the access table. If no other station is accessing the file, see the ISS Utility FILE STATUS REPORT to clear the access table.

Table 34-5. Numeric List of SORT-4 Error Messages and Recovery Procedures (Cont'd)

98. ERROR MESSAGE NOT DEFINED

Following any error message, the program displays:

STOP ERROR ENDING

99. RECORD COUNT XXXXX
STOP END OF SORT

This is the normal ending. If error messages are passed to the next program, the value S9 = 99 indicates that the sort was executed with no errors.

34.8 SORT-4 TIMINGS

In general, SORT-4 takes longer for the set-up phase (1 to 2 minutes) than SORT-3 (about 30 seconds), because SORT-4 has more options to choose from. But the actual sorting time is less in SORT-4 than SORT-3. If the file is large enough to make up for the increased set-up time, SORT-4 will run faster than SORT-3.

The table below gives some comparisons of running time between SORT-4 and SORT-3. The SORT-3 times were taken from actual timings of the 2200 Commercial Matrix Disk Sort, which was revised slightly to become the ISS Disk Sort Utility, which was revised again to become SORT-3. As representative ("ballpart") timings of SORT-3, they are probably accurate to within 15 seconds. The SORT-4 times are actual timings of SORT-4, and are thus representative of SORT-4 capabilities only under the prescribed conditions.

In the table below:

File 24/24 consists of 24-byte records with a 24-byte sort key. The record is all one field. There are 10 records per block.

File 120/8 consists of 120-byte records with an 8-byte (numeric) sort key. The record contains 6 fields. There are 2 records per block.

File 120/64 is the same file as above, except that 4 fields (1 numeric and 3 alpha) or a total of 64 bytes, are included in the sort key.

Sort type "R" is a full record sort, and "K" is a key sort, and "T" is a tag sort.

Input records are in random order. A 5-megabyte hard disk is used in all cases. All sort times are in minutes, and will vary depending on the number of fields in a record, the amount of memory available and other factors.

MEMORY SIZE	FILE	RECORDS	SORT TYPE	SORT-3 TIME	SORT-4 TIME
8K	24/24	2000	R	8.22	7.75
8K	24/24	20000	R	92.0	81.7
8K	120/8	1000	K	7.65	7.9
8K	120/64	1000	K	12.25	12.05
32K	24/24	2000	R	7.0	6.4
32K	24/24	8000	R	27.3	23.6
32K	120/8	1000	K	7.3	7.7
32K	120/8	4000	K	28.8	27.3
32K	120/64	1000	R	9.22	8.9

SORT-4 TIMINGS, 2200VP

MEMORY SIZE	FILE	RECORDS	SORT TYPE	SORT-4 TIME 2200VP
16K	24/24	2000	R	1.0
16K	24/24	20000	R	9.1
16K	120/8	4000	K	9.2
16K	120/8	4000	R	9.55
16K	120/8	4000	T	2.5
16K	120/64	1000	K	3.0
16K	120/64	1000	R	2.3
64K	24/24	20000	R	6.8
64K	120/8	4000	K	9.0
64K	120/8	4000	R	6.9
64K	120/8	4000	T	2.5
64K	120/64	1000	R	1.8
64K	120/64	4000	R	7.0

APPENDIX A - KFAM UTILITY ERROR MESSAGES AND RECOVERY PROCEDURES

KFAM Utility Programs provide many error messages which indicate different conditions. In general, these error messages fall into one of three error categories, as listed under DISPLAY CONDITION below in Table A-1.

Display Condition A indicates that a prompt and its associated entry field accompany the recoverable error message, allowing continuation of the program in progress. Display Condition B indicates an non-recoverable error message, is rarely encountered, and requires the operator to touch Special Function (S.F.) Key 31 to close all system and KFAM files. Display Condition C indicates an non-recoverable error and a STOP condition. If "STOP" is displayed, the files are closed. Display Conditions B and C usually require program reload, as well as other procedures.

Table A-2, which follows Table A-1, provides specific recovery procedures for each possible error message.

Special 2200MVP Considerations

The 2200MVP partition configuration executed may specify that one terminal number is assigned to multiple partitions. If an error occurs, the corresponding error message appears immediately on the assigned terminal's screen only if the terminal is currently attached to the partition which encountered the error. Otherwise, the error message is displayed when the terminal becomes attached to the partition encountering the error, e.g., by means of a $RELEASE TERMINAL statement.

NOTE:

KFAM-7 hogs program execution of portions of global text at critical times for a particular station number. Should a program's execution be aborted during this "program hog", the "program hog" flag, global variable @T, is left equal to the station number which causes all other program execution to "hang". To correct this "hanging" condition, complete the following steps from one of the "hanging" stations: (1) touch the HALT key, (2) enter PRINT @T (immediate mode) to determine the station number causing the hanging condition, (3) check with the user at the station number causing the "hanging" condition, (4) enter @T=0 to clear the program hog flag, and (5) touch the CONTINUE key and the RETURN key to automatically cause normal operation to resume.

Table A-1. Error Message Categories and Recovery Options

DISPLAY CONDITION	RECOVERY OPTIONS
A. Error messsage appears with prompt.	A-1. The operator usually re-enters the requested information *and* then continues with the next step according to the program being run. A-2. To escape from (abort) this program, touch S.F. Key 31 once to close all files and obtain the KFAM-7 menu.
B. Error message appears without any prompt, and "STOP" is <u>not</u> displayed (this is rare).	B. Touch S.F. Key 31 once to close all files. Refer to RECOVERY OPTIONS below for Condition C.
C. "STOP" appears on screen, or Recovery Option A-2 or B just completed. ("STOP" is usually accompanied by an error message and indicates that all files are closed.)	C-1. If this program was loaded from the KFAM menu, the operator may load a KFAM utility by touching S.F. Key 31 to bring the KFAM menu to the screen. C-2. Otherwise, the operator may touch CLEAR and RETURN and then LOAD a program.

NOTE:

If the error recovery procedures require S.F. Key 31 to be touched, Edit mode (blinking cursor) must be switched off (steady cursor) before S.F. Key 31 is touched. Edit mode is manually switchable by means of the EDIT key.

Table A-2. KFAM Utility Error Messages

All KFAM Utility error messages are listed below in alphabetical order. For general recovery procedures, refer to Table A-1 before attempting recovery from any error message.

ERROR MESSAGE	DESCRIPTION	RECOVERY
ACCESS ERROR RESTORE BOTH USER FILE AND KEY FILE FROM BACKUP COPIES BEFORE ATTEMPTING TO RE-RUN THIS PROGRAM (STOP).	Could be due to no records in the User File. Also, could be a machine error or Key File problem. The partially copied User File and Key File are partially reorganized and are thus destroyed.	See Table A-1. Copy the backup User File and Key File, as partially reorganized files are destroyed. After copying, run KEY FILE RECOVERY. Rerun this utility.
ANY ERROR DURING THE RUNNING OF KFAM3207 WILL DESTROY BOTH FILES. MAKE COPIES OF THE DISK PLATTERS CONTAINING THE USER FILE AND KEY FILE BEFORE RUNNING THIS PROGRAM.	Running this program requires that back-up copies were previously made.	See Table A-1. Make backup copies of the User File and Key File. Then rerun this utility.
BLOCKING FACTOR OR RECORD LENGTH INCORRECT	Record length times blocking factor, plus all control bytes (DC, DA access only), must not exceed 256. See Section 19.1 for further explanation; applies to record types A, B, C.	Recalculate record length, blocking factor, if required. Re-enter LOGICAL RECORD LENGTH and continue.
DUPLICATE KEY IGNORED (printed)	Duplicate keys, when encountered, are excluded from the KFAM File. Their (hex) relative sector location is also printed as it exists in User File, but the key isn't entered in the Key File. Record number also appears with pointer.	Execution error, no operator action is required unless ISS printer address blank; if blank, key CONTINUE and RETURN to resume. Erroneous key and its record should be corrected and later added to file.

END NOT DEFINED	An END record does not exist for this User File.	Enter a reply to ENTER LAST KEY and continue. If RETURN was just entered, this error indicates that no END record found, or all deleted records not flagged with hex FF in first byte of key, thus the last key's value must be entered.
ERROR OPENING FILES (STOP)	An error condition other than access mode conflict was encountered while attempting to open KFAM files.	See Table A-1. Rerun this program. If this error message persists, contact Wang Laboratories, Inc.
ERR 1nn	Indicates an error as described below for certain conditions. Note that if a statement line beginning with an "at sign" (@) appears, the error was encountered during global program text execution and may be caused by an application program error or an erroneous Key Directory Record (KDR) in the Key File.	See Table A-1 and the errors listed below for certain conditions; refer to the Disk Reference Manual or the BASIC-2 Language Reference Manual for other errors.

ERR I96	Disk read error (sector cannot be read). In program statement, #1 displayed indicates sector contained in Key File. #2 displayed indicates User File sector, (except for REORGANIZE SUBSYSTEM where #1 indicates User File and #2 indicates Key File). For all utilities, #0 indicates sector on KFAM system disk, #T1 or T1(T9) indicates sector in Key File. This indicates which disk may be defective. Also, could be a hardware error, especially if operating environment limits are exceeded (temperature and humidity).	Note the file number, then touch S.F.Key 31 to close file. See Table A-1. In general, restore same disk or create new disk from backup copy. Rerun the utility. With REORGANIZE SUBSYSTEM, run KEY FILE RECOVERY on input User File. If Part 3, reorganize output User File, assigning its name as the input User File name, and then rerun REORGANIZE SUBSYSTEM. If error persists, file is permanently damaged, or hardware malfunction has occurred.
ERR I99	Disk write error (sector cannot be written). Most likely caused by a bad physical sector. In program statement, #1 displayed indicates Key File sector. #2 displayed indicates User File sector (except for REORGANIZE SUBSYSTEM where #1 indicates User File and #2 indicates Key File).	Record the file number, and depress S.F. Key 31 to touch files. See Table A-1. Restore backup copy to new disk; the old disk is not useable for this file and should be discarded (after all files are copied from it if not on backup). Rerun this program. With REORGANIZE

	For all utilities, #0 indicates sector on KFAM system disk; #T1 or #T1(T9) indicates Key File sector. This indicates which disk may be defective.	SUBSYSTEM, Part 1: Output User File contains a bad physical sector. Part 2: Output Key File contains a bad physical sector. Part 3: Input User File contains a bad physical sector. RECOVERY: Part 1: Replace the output disk or recreate file to bypass the bad sector. If input and output Key File are the same, run Key File Recovery. Rerun. Part 3: Replace input disk or recreate input User File to bypass the bad sector. See recovery procedure for ERR I96, Part 3.
ERR X74	A KFAM file which is not a KFAM-7 file was accessed, e.g., KFAM-3, KFAM-4 file.	Touch S.F. Key 31 to close files, then see Table A-1. Either use a CONVERT KFAM FILE utility to convert this to KFAM-7 format and then rerun this program or if wrong file name, disk address, or disk on-line, rerun this program taking care correct file is accessed.
ERROR#XX LINE XXXX (also ERR XX)	The error code and line number are displayed. Refer to typical ERR 1nn codes listed above.	See Table A-1. Rerun this program. If this error persists, notify Wang Laboratories, Inc.

FILE ALREADY CATALOGED	The User File or the Key File, whose file name was created from the User File name and Key File number, already exists at the specified disk address. Prompt indicates which file was already cataloged.	Re-enter USER FILE NAME, or re-enter KEY FILE NUMBER, depending on which file is already cataloged.
FILE XXXXXXXX ALREADY CATALOGED ON DEVICE XYY	A file designated as "not cataloged" is already cataloged, or a file with the same name exists at the specified disk address. (File name and address are displayed.)	See Table A-1. Mount a scratch disk or a disk that does not have this file cataloged and rerun this program. If this error recurs, change the values of 03$ and 04$ (user set-up module) to "C" and rerun.
FILE NOT AVAILABLE	The KFAM file is currently being accessed by another station whose access mode conflicts with that required by this KFAM utility. For all utilities but PRINT KEY FILE, exclusive access is required. For PRINT KEY FILE, Read Only is required. Exclusive access requires that no other stations are accessing the file. Read Only requires no stations in Shared or Exclusive access modes.	See Table A-1. Rerun this program. If unsuccessful after several reentries, run PRINT KEY FILE to determine if RESET ACCESS TABLE is required. Run RESET ACCESS TABLE if file accidentally left open. Rerun this program.

FILE NOT FOUND	The specified User File or Key File could not be located at the specified disk address. The file that could not be located is indicated by the prompt accompanying this error message.	If User File not found, re-enter USER FILE NAME and continue. If Key File not found, re-enter KEY FILE NUMBER and continue. If this error persists, check if correct disk is at the specified disk address.
FILE XXXXXXXX NOT FOUND ON DEVICE XYY	A file designated as "cataloged" is not cataloged at the specified disk address. (File name and disk address are displayed.)	See Table A-1. Mount the correct disk at appropriate address and rerun this program. Enter correct file name if incorrect.
FINDFIRST ERROR	Hardware or software error.	See Table A-1. Rerun this program. If this error persists, contact Wang Laboratories, Inc.
FINDNEXT ERROR	Hardware or software error.	See Table A-1. Rerun this program. If this error persists, contact Wang Laboratories, Inc.
INPUT AND OUTPUT USER FILE MAY NOT BE THE SAME FILE	Both input and output User Files are designated by the same file name at the same disk address.	See Table A-1. Correct the file name designations within program (user set-up module). Rerun this program.
INPUT FILE NOT AVAILABLE	The requested input file is currently being accessed by another station (files are opened in Exclusive access mode).	See Table A-1. Rerun, or wait and rerun. If this error persists, use PRINT KEY FILE to list access table. If the file was accidentally left open, run RESET ACCESS TABLE, then rerun this program.

INSUFFICIENT SPACE FOR FILE XXXXXXXX ON DEVICE XYY	There is not enough disk space on the designated disk device to catalog the file.	See Table A-1. Mount an output disk with enough space to accomodate the output User File and/or Key File. Rerun this program.
INVALID	The Key File number entered is not a digit from 1-9.	Re-enter KEY FILE NUMBER and continue.
INVALID DELIMITER	Hardware or software error.	See Table A-1. Rerun this program. If this error persists, contact Wang Laboratories, Inc.
INVALID DEVICE ADDRESS	The xyy form of the disk device address is not a valid disk address.	Re-enter the DEVICE ADDRESS for the User File or Key File (indicated in the prompt) and continue.
INVALID KEY FILE NUMBER	The Key File number was not a number from 1-9 or not an integer.	See Table A-1. Correct Key File number in the user set-up module (program). Rerun this program.
INVALID KEY RESTORE BOTH USER FILE AND KEY FILE FROM BACKUP COPIES BEFORE ATTEMPTING TO RUN THIS PROGRAM	Keys within the Key File do not match keys within User File records, or active key in Key File is flagged as deleted in User File, or record length (blocked records) specified wrong, or starting position of key specified wrong. Could be caused by an application program error.	See Table A-1. Attempt to determine the problem (one of the items listed under DESCRIPTION) by completing the recovery procedures described for SEQUENCE ERROR (REORGANIZE IN PLACE) below (e.g., write a program to compare keys, or to correct record length, key position in KDR).

INVALID KEY, HEX VALUE= XXXXXXX... KEY RETURN (EXEC) TO SKIP RECORD	The hex value of the invalid key is displayed, which differs from the value of the key in the Key File. The record is active in the Key File, but flagged as deleted in the User File.	To skip this record, key RETURN. Otherwise, touch S.F. Key 31 to abort this. program. Check for errors in application programs that may have caused this condition. Run KEY FILE RECOVERY on input User File. Rerun this program. If unsuccessful, see INVALID KEY above. Rerun this program.
INVALID--KEY MUST BE 2 TO 30	The entered value for key length must be between 2 and 30 (inclusive).	Re-enter the KEY LENGTH with a correct value and continue.
INVALID - MUST BE 2 TO 255	The value for the number of sectors per record must be between 2 and 255 (inclusive) for type M records.	Re-enter NUMBER OF SECTORS PER RECORD and continue.
INVALID PASSWORD (appears with prompt)	The Password entered for this User File is not identical to the Password previously assigned to this file. May be caused by entering a Password where blanks are required.	Re-enter the PASSWORD, if one is required, and continue.
INVALID PASSWORD (STOP)	The Password entered for this User File is incorrect. The Password entered must be identical to the one assigned to this file upon creation.	See Table A-1. Rerun this program, taking special care in entering the Password. Repeated attempts met with failure may indicate wrong User File name being entered or wrong disk on-line.
INVALID PASSWORD, INPUT	The Password specified for the input User File is incorrect (does not match Password previously assigned to this User File).	See Table A-1. Check value of P$; if wrong, correct value of P$. Rerun this program.

INVALID PASSWORD, OUTPUT	The Password specified for the previously cataloged output User File is incorrect (does not match Password previously assigned this User File).	See Table A-1. Check value of P9$; if wrong, correct value of P9$. Rerun this program.
INVALID POINTER	Sector accessed is outside of User File boundaries. Probably the "END" record does not contain the information necessary to build/rebuild the Key File.	See Table A-1 Rerun this program. If this error persists, contact Wang Laboratories, Inc. (recovery may not be possible).
INVALID RECORD TYPE	The entry made for record type was invalid. Valid entries include A,B,C,M, and N.	Re-enter the RECORD TYPE and continue.
INVALID RECORD FORMAT	Applies to Type A records; either more than one record per sector, more than 38 fields per record, or record written without the correct control bytes. End record may be invalid.	See Table A-1. With KEY FILE CREATION or REORGANIZE IN PLACE, this utility cannot be run until User File records are rewritten (file is re-created). With KEY FILE RECOVERY, run INITIALIZE KFAM FILE and BUILD KEY FILE, then rerun. With REORGANIZE SUBSYSTEM, the applications programmer should re-create the file if the User File is wrong or re-create file parameters in the KDR (see Section 27.5). Then rerun.

INVALID STATION NUMBER	The station number designated is invalid.	See Table A-1. Change the station number (S2) to a value 1-16 in the set-up module (program). Rerun.
KEY FIELD OUT OF BOUNDS	Applies to record type A: the key must be wholly contained within one field. End record may be invalid.	See Table A-1. Check key length, starting position using PRINT KEY FILE. Then refer to recovery procedures described for INVALID RECORD FORMAT for the program in use.
KEY FILE SPACE EXCEEDED RESTORE BOTH USER FILE AND KEY FILE FROM BACK-UP COPIES BEFORE ATTEMPTING TO RE-RUN THIS PROGRAM (STOP)	Allocated space in the Key File is exhausted, and both User File and Key File are partially reorganized, and thus destroyed.	See Table A-1. Copy User File and Key File from backup copies; with Key File, increase space allocated by increasing extra sectors with COPY/VERIFY. Run REALLOCATE KFAM FILE SPACE. Rerun this program.
KEY MAY NOT SPAN SECTORS	The key location, as specified by the starting location, will span two sectors and is thus invalid. Applies only to M type records.	Recalculate the starting location or length of the key. Re-enter the STARTING POSITION OF KEY Length and continue.
KEY OVERLAPS END RECORD	The key goes beyond the boundaries of the record, as determined by the record type and record length.	Recalculate the starting position of the key. Re-enter the STARTING POSITION OF KEY and continue.

KFAM-4 FILE BUSY	The specified KFAM-4 input file is currently being accessed by another station.	See Table A-1. Rerun this program. If this error persists, run KFAM-4 version of PRINT KEY FILE to determine the station (or CPU) accessing the file using a 2200T or 2200VP Central Processor to access the disk(ette) containing the KFAM-4 files; if the file was accidentally left open, run KFAM-4 version of RESET ACCESS TABLE. Rerun this program.
LAST KEY NOT FOUND RESTORE BOTH USER FILE AND KEY FILE FROM BACKUP COPIES BEFORE ATTEMPTING TO RE-RUN THIS PROGRAM (STOP)	REORGANIZE IN PLACE: the User File and Key File are partially reorganized and thus are undefined. The last key in the Key File could not be located in the User File, or vice versa.	See Table A-1. Copy the backup Key File and User File as the partially unorganized files are undefined. The applications programmer should write a small program that will determine the values of the two keys that do not match. Following FINDFIRST or FINDNEXT, T7$ contains the key value from the Key File. This can be compared to the corresponding key value in the User File. The non-matching keys should be corrected or deleted. Then rerun this program.

LAST KEY NOT FOUND	KEY FILE CREATION: The value of the key entered as the last key does not match the actual value of the last key.	See Table A-1. Run INITIALIZE KFAM FILE. Then rerun KEY FILE CREATION and take care in entering the last key, if required.
MORE THAN 40 SECTORS PER RECORD	Applies to type M records. This program will not reorganize a file whose records exceed 40 sectors in length.	Touch S.F. Key 31, then see Table A-1. Use REORGANIZE SUBSYSTEM instead of REORGANIZE IN PLACE.
NO ROOM ON DISK FOR OUTPUT PROGRAM	There is not enough room on the disk for the output program to be cataloged.	See Table A-1. Rerun this utility, and mount a disk with enough space to accommodate the output program (52 sectors maximum requirement).
NO SPACE	Not sufficient space for Key File. Possibly last key was incorrectly entered.	See Table A-1. With KEY FILE CREATION, run INITIALIZE KFAM FILE, then rerun this program. With KEY FILE RECOVERY, run COPY/VERIFY on Key File increasing extra sectors value; then run REALLOCATE KFAM FILE SPACE. Rerun this program.
NO SPACE ON DISK FOR KEY FILE	There is insufficient space on this disk to catalog the Key File.	See Table A-1. With KEY FILE CREATION, run INITIALIZE KFAM FILE, then rerun this program. With KEY FILE RECOVERY, mount a disk with enough free space to accommodate the Key File. Rerun this program.

NOT BLOCKED AS SPECIFIED	Record type A: records per block specified incorrectly, or records not written in array format.	See Table A-1. With KEY FILE CREATION or REORGANIZE IN PLACE, this utility cannot be run until User File records are rewritten (file is re-created). With KEY FILE RECOVERY, run INITIALIZE KFAM FILE and KEY FILE CREATION, then rerun. With REORGANIZE SUBSYSTEM, the application programmer should re-create the User File if it is wrong, or re-create file parameters in the KDR (see Section 27.5). Then rerun.
NOT DATA FILE	The User File specified is a program file, whereas only data files are valid as KFAM User Files. Either the User File Name or the User File device address is invalid, or the wrong disk is on-line.	Re-enter User File name. If unsuccessful again, mount correct disk if wrong disk is on-line, check if disk address entered is correct, and rerun.
NOT KFAM FILE NAME	The User File name entered does not conform to KFAM file name conventions, which require that (1) an "F" must be in position 5 and (2) a digit 0-9 must be in position 6.	Re-enter User File name according to KFAM naming conventions (SSSSFNSS).
NULL FILE	There are no active records in this KFAM file.	See Table A-1. Probably wrong KFAM file name entered; if so, rerun with correct file name.

NUMERIC KEY INVALID	Type A records: The key field is indicated as lying within a numeric variable. The key may not be a numeric variable.	See Table A-1. Recovery procedures same as for NOT BLOCKED AS SPECIFIED (see above).
OPERATOR INTERRUPT	Program was interrupted by the operator depressing S.F. Key 31.	See Table A-1. Rerun this program.
OUTPUT FILE NOT AVAILABLE	The specified output file is currently being accessed by another station and is therefore not available at this time.	See Table A-1. Rerun this program. If this error persists, rerun PRINT KEY FILE and determine if the file was accidentally left open, run RESET ACCESS TABLE, and then rerun this program.
OUTPUT KEY FILE SPACE EXCEEDED	Output Key File is too small.	See Table A-1. If output Key File is the same as the input Key File, then run KEY FILE RECOVERY on input User File. Allocate more cataloged space for output Key File. Rerun this program.
OUTPUT PROGRAM SPACE EXCEEDED	Output program too big for allocated file space (file cataloged successfully).	See Table A-1. Reselect BUILD SUBROUTINE MODULE from KFAM menu. During rerun, either assign a different program file name, or mount a disk that does not contain this file name (maximum 52 sectors required).

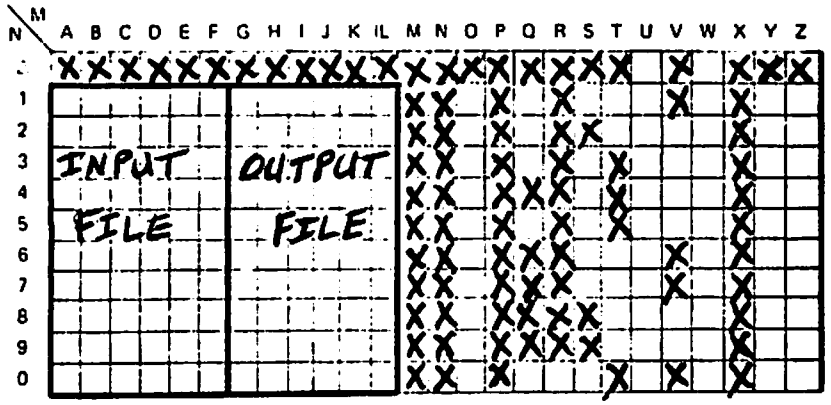
OUTPUT USER FILE SPACE EXCEEDED	Space allocated for output User File in set-up module is too small.	See Table A-1. Correct set-up module program. Either let REORGANIZE SUBSYSTEM catalog a new file, or manually catalog a new output User File. Rerun.
PRINTING ERROR REPORT	Appears while an error report is being printed.	No operator action is necessary. The error report should be examined after printing.
RECORD LENGTH NOT SPECIFIED CORRECTLY	Type A records: record length specified in INITIALIZE KFAM FILE does not equal record length of actual record.	See Table A-1. Recovery procedures are the same as for NOT BLOCKED AS SPECIFIED (see above).
RE-ENTER	Entry made is invalid because it contained too many characters, a "Y" or "N" was not entered in reply to a YES/NO question, numeric/alphanumeric field conflicts with entry made, entry outside of range for this field, etc.	Re-enter a reply to the same prompt field.
RESTORE BOTH USER FILE AND KEY FILE FROM BACK-UP COPIES BEFORE ATTEMPTING TO RERUN THIS PROGRAM	Refer to other message displayed with this error message, which is listed elsewhere in this table in alphabetical order.	See accompanying message for specific recovery procedures. This indicates KFAM files are destroyed and backup copies are required.
SECTORS AVAILABLE, DEVICE XYY... SECTORS REQUESTED, DEVICE XYY...	There is not enough room on this disk to catalog Key File and/or User File. (Device, sectors available, sectors requested are displayed.)	See Table A-1. Mount a different disk with enough free sectors to accommodate both files. Rerun this program.

SEQUENCE ERROR	REORGANIZE SUBSYSTEM: indicates the key contained in the input User File does not match the key contained in the input Key File (KDR).	See Table A-1. Run KEY FILE RECOVERY on User File. Rerun.
SEQUENCE ERROR RESTORE BOTH USER FILE AND KEY FILE FROM BACKUP COPIES BEFORE ATTEMPTING TO RE-RUN THIS PROGRAM.	REORGANIZE IN PLACE: indicates the keys in the User File do not match the keys in the Key File (KDR). Could also be machine error. Both User File and Key File are partially re-organized and thus are effectively destroyed.	See Table A-1. Copy backup copies of User File and Key File. The applications programmer should write a small program to determine which keys do not match. Following FINDFIRST or FINDNEXT, T7$ contains the key value from the Key File. This can be compared to the corresponding key in the User File. The non-matching keys should be corrected or deleted; then, rerun this program.
STOP NO ROOM FOR KEY FILE	The User File is already cataloged. There is insufficient space on this disk to catalog the Key File.	See Table A-1. Mount a different disk with enough free space to accommodate the Key File. Rerun this program.
STOP WORK FILE FULL	The work file KFAMWORK is full and cannot contain aditional error messages to be printed.	Refer to Table A-1. Run the ISS utility Disk Dump to print the contents of KFAMWORK on the KFAM system disk.

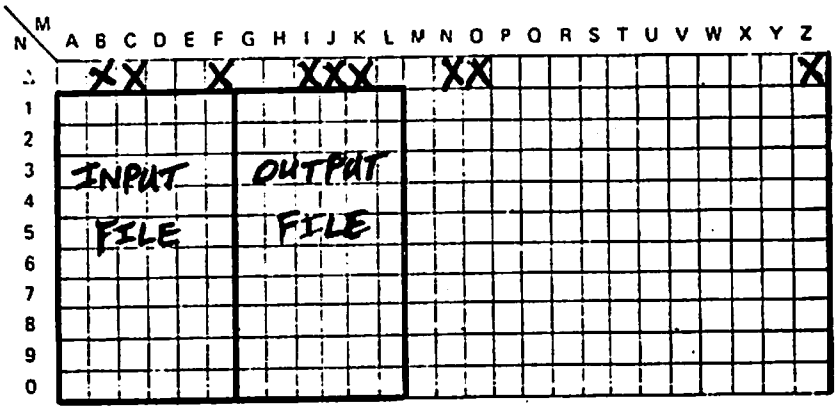
SYSTEM ERROR	Hardware or software error.	Touch S.F. Key 31 and see Table A-1. Rerun this program. Notify Wang Laboratories, Inc. if this error persists.
UNREADABLE SECTOR NNNNN, NNN RECORDS LOST (printed)	A sector of the User File cannot be read. The sector number and number of records lost due to the unreadable sector are displayed. This message replaces ERR I96 for KEY FILE RECOVERY only.	Execution error, no operator action required unless printer address is blank; if blank, key CONTINUE and RETURN to resume.
USER FILE TOO SMALL	The User File, which is already cataloged, does not have enough room for the estimated number of records.	Re-enter ESTIMATED NUMBER OF RECORDS with a smaller number and continue. After completion of this program, use COPY/VERIFY to increase the allocation for this file; then run REALLOCATE KFAM FILE SPACE. After completion, the file is available for use by other software.
WAITING FOR PRINTER	The printer is not ON and SELECTED, or the printer is currently being used by another station.	Ready the printer if it is not ON and SELECTED, or wait until it becomes available. When ready and available, printing automatically begins and continues without operator intervention.

WARNING - KEY FILE IS TOO SMALL (also printed)	The already cataloged Key File is too small to accommodate the estimated number of records. This is only a warning.	Either let program continue until Key File is full, then REORGANIZE, or the program may be stopped by touching S.F. Key 31, then run ISS COPY/VERIFY with enough EXTRA SECTORS to increase allocation of Key File; then run REALLOCATE KFAM FILE SPACE. Rerun this program.
WORK FILE NOT AVAILABLE	Work file KFAMWORK is either currently being accessed by another station or does not exist.	See Table A-1. Rerun this program. If this error persists, check if other stations are running KFAM utility programs. If other stations are running KFAM utility programs, keep trying or wait until they are done. If no other stations are running KFAM utility programs, run ISS FILE STATUS REPORT Utility to obtain the file access status of "KFAMWORK" and, if the file was accidentally left open, either use FILE STATUS REPORT to close "KFAMWORK" or execute a non-KFAM file Close sub-routine to close "KFAMWORK". Rerun this program.
8 LEVELS OF INDEX EXCEEDED	More than 390,625 30-byte keys, or more than 429,981,696 12-byte keys, etc. This error should not occur.	See Table A-1. Run PRINT KEY FILE for the Key File in use. Examine the printed report and notify Wang Laboratories, Inc.

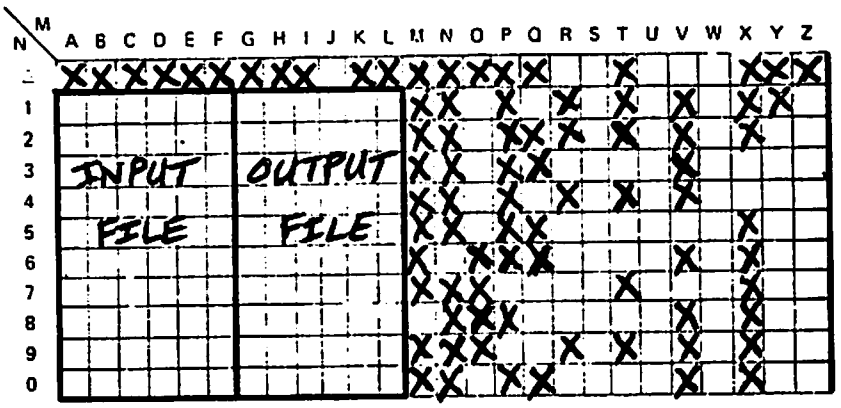
APPENDIX B - SORT-4 VARIABLE CHECK OFF LIST



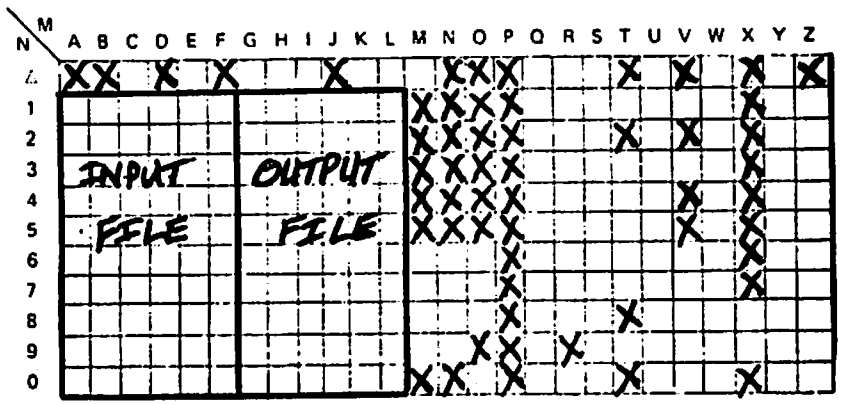
NUMERIC SCALARS
FORMAT = MN



NUMERIC ARRAYS
FORMAT = MN(



ALPHA NUMERIC SCALARS
FORMAT = MN$



ALPHA NUMERIC ARRAYS
FORMAT = MN$(

APPENDIX C - CONDITIONS GOVERNING SPURIOUS RESULTS FROM THE LIST/CROSS-REFERENCE UTILITY

Under certain conditions the variable cross-reference table printed by the LIST/CROSS-REFERENCE utility may be erroneous. Specifically, certain BASIC-2 statements in the input program can cause nonvariables to be referenced as variables. A second condition causes array variables to be referenced as scalar variables. A third condition occurs where variables are not referenced.

Because BASIC-2 is a flexible high-level language, it is not possible to check each statement's possible syntax as with Assembler Languages. Instead, the Cross-Reference checks the current and the previous byte, and looks in a table for possible variable and nonvariable values as it reads each statement line.

The BASIC-2 statements and accompanying conditions for these errors are given below. Also, please note that the ISS-3.7 LIST/CROSS-REFERENCE does <u>not</u> support statements (e.g., multi-programming statements) or variables (global variables) applicable only to the 2200MVP.

Non-Variables Referenced as Variables by List/Cross Reference

STATEMENT	CONDITION AND VARIABLE POSITION
PLOT	All D, U, C, S, and R pen control characters.
DATASAVE BT	The "N" of the "N" parameter specifying block size. The H parameter specifying the header block mark. The R which specifies resave.
DATALOAD BT	The "N" of the "N" parameter specifying block size.
ADD, ADD C, AND, OR, XOR, BOOL, INIT, $TRAN, POS,	In these statements, if hexadecimal digits (X_1X_2) appear where X_1 = A, B, C...F and X_2 = 0, 1, 2...9, then the hexadecimal digits are referenced as if they collectively represent a variable. If X =0,1,2,...9, then these are not interpreted as a variable. Examples include: AND (A$,XX) and TRANS(A$,B$) [XX] [R]

SAVE, LOAD, MOVE, COPY (DC followed by platter parameter)	The platter parameter (F,R,T) is interpreted as a variable if followed by a slash "/" or parentheses "(". For example, LOAD DC F/B10, "FILE" and SAVE DC F(), "FILE" F/B10 cause F to be referenced as a variable.
SAVE <S> F "FILE"	In this example of the SAVE (BASIC-2) statement, S is interpreted as a variable.
$PACK, $UNPACK	If F or D follows $PACK or $UNPACK, the F or D is interpreted as a variable.
DSKIP( )S, DBACKSPACE( )S	References S as a variable.
SKIP( )F, BACKSPACE( ) F	References F as a variable.

Array Variables Referenced as Scalar Variables

STATEMENT	CONDITION AND VARIABLE POSITION
All the MATH MATRIX Statements	All the array variables which appear in these statements are referenced as if they are scalar variables, unless a variable is followed by a left parenthesis. For example, MAT A=B is interpreted as if A and B both were scalars; whereas MAT A=B(4,64) is interpreted as if A is a scalar and B is an array.

Variables Not Referenced By List/Cross Reference

STATEMENT	CONDITION AND VARIABLE POSITION
SCRATCH followed by platter parameter	The first variable following the platter parameter is not interpreted as a variable. For example, the statement SCRATCH F, A$, B$, C$ is interpreted A$ were not a variable, however, B$ and C$ are interpreted as variables.
All statements applicable only to the 2200MVP.	Always.
All global variables.	Always.

To help us to provide you with the best manuals possible, please make your comments and suggestions concerning this publication on the form below. Then detach, fold, tape closed and mail to us. All comments and suggestions become the property of Wang Laboratories, Inc. For a reply, be sure to include your name and address. Your cooperation is appreciated.

700-4768

TITLE OF MANUAL: **INTEGRATED SUPPORT SYSTEM (ISS) RELEASE 3.7 USER MANUAL**

COMMENTS:

Fold

Fold

(Please tape. Postal regulations prohibit the use of staples.)

WANG

Fold

FIRST CLASS
PERMIT NO. 16
Tewksbury, Mass.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

– POSTAGE WILL BE PAID BY –

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Attention: Technical Writing Department

Fold

Cut along dotted line.

Printed in U.S.A.
13-1019

North America:

Alabama
Birmingham
Mobile

Alaska
Anchorage

Arizona
Phoenix
Tucson

California
Fresno
Inglewood
Los Angeles
Sacramento
San Diego
San Francisco
San Mateo
Sunnyvale
Tustin
Ventura

Colorado
Denver

Connecticut
New Haven
Stamford
Wethersfield

District of Columbia
Washington

Florida
Jacksonville
Miami
Orlando
Tampa

Georgia
Atlanta

Hawaii
Honolulu

Illinois
Chicago
Morton
Park Ridge
Rock Island

Indiana
Indianapolis
South Bend

Kansas
Overland Park
Wichita

Kentucky
Louisville

Louisiana
Baton Rouge
Metairie

Maryland
Rockville
Towson

Massachusetts
Boston
Burlington
Littleton
Lowell
Tewksbury
Worcester

Michigan
Grand Rapids
Okemos
Southfield

Minnesota
Eden Prairie

Missouri
Creve Coeur

Nebraska
Omaha

Nevada
Reno

New Hampshire
East Derry
Manchester

New Jersey
Howell
Mountainside

New Mexico
Albuquerque

New York
Albany
Buffalo
Lake Success
New York City
Rochester
Syracuse

North Carolina
Charlotte
Greensboro
Raleigh

Ohio
Cincinnati
Columbus
Middleburg Heights
Toledo

Oklahoma
Oklahoma City
Tulsa

Oregon
Beaverton
Eugene

Pennsylvania
Allentown
Camp Hill
Erie
Philadelphia
Pittsburgh
Wayne

Rhode Island
Cranston

South Carolina
Charleston
Columbia

Tennessee
Chattanooga
Knoxville
Memphis
Nashville

Texas
Austin
Dallas
Houston
San Antonio

Utah
Salt Lake City

Virginia
Newport News
Richmond

Washington
Seattle
Spokane

Wisconsin
Brookfield
Madison
Milwaukee

Canada
Wang Laboratories (Canada) Ltd.
Don Mills, Ontario
Calgary, Alberta
Edmonton, Alberta
Winnipeg, Manitoba
Ottawa, Ontario
Montreal, Quebec
Burnaby, B.C.

International Subsidiaries:

Australia
Wang Computer Pty. Ltd.
Sydney, NSW
Melbourne, Vic.
Canberra, A.C.T.
Brisbane, Qld.
Adelaide, S.A.
Perth, W.A.
Darwin, N.T.

Austria
Wang Gesellschaft M.B.H.
Vienna

Belgium
Wang Europe, S.A.
Brussels
Erpe-Mere

Brazil
Wang do Brasil Computadores Ltda.
Rio de Janeiro
Sao Paulo

China
Wang Industrial Co., Ltd.
Taipei, Taiwan

France
Wang France S.A.R.L.
Bagnolet
Ecully
Nantes
Toulouse

Great Britain
Wang Electronics Ltd.
Northwood Hills, Middlesex
Northwood, Middlesex
Harrogate, Yorkshire
Glasgow, Scotland
Uxbridge, Middlesex

Hong Kong
Wang Pacific Ltd.
Hong Kong

Japan
Wang Computer Ltd.
Tokyo

Netherlands
Wang Nederland B.V.
Ijsselstein

New Zealand
Wang Computer Ltd.
Grey Lynn, Auckland

Panama
Wang de Panama (CPEC) S.A.
Panama

Republic of Singapore
Wang Computer Pte., Ltd.
Singapore

Republic of South Africa
Wang Computers (South Africa) (Pty.) Ltd.
Bordeaux, Transvaal
Durban
Capetown

Swedan
Wang Skandinaviska AB
Solna
Gothenburg
Arloev
Vasteras

Switzerland
Wang S.A./A.G.
Zurich
Bern
Pully

West Germany
Wang Laboratories GmbH
Berlin
Cologne
Duesseldorf
Fellbach
Frankfurt/M.
Freiburg/Brsg.
Hamburg
Hannover
Kassel
Munich
Nuernberg
Stuttgart

International Representatives:

Argentina
Bolivia
Canary Islands
Chile
Colombia
Costa Rica
Cyprus
Denmark
Dominican Republic
Ecuador
Finland
Ghana
Greece
Guatemala
Iceland
India
Indonesia
Iran
Ireland
Israel
Italy
Jamaica
Japan
Jordan
Kenya
Korea
Lebanon
Liberia
Malaysia
Mexico
Morocco
Nicaragua
Nigeria
Norway
Pakistan
Peru
Philippines
Portugal
Saudi Arabia
Spain
Sri Lanka
Syria
Thailand
Tunisia
Turkey
United Arab Emirates
Venezuela
Yugoslavia

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 851-4111, TWX 710 343-6769, TELEX 94-7421

Printed in U.S.A.
700-4768
6-78-5C
Price: see current list