WANG

# Integrated Support System (ISS) Release 5 User Manual

# INTEGRATED SUPPORT SYSTEM (ISS) RELEASE 5 USER MANUAL

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 459-5000, TWX 710 343-6769, TELEX 94-7421

WANG LABORATORIES, INC.

ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 459-5000, TWX 710 343-6769, TELEX 94-7421

## HOW TO USE THIS MANUAL

The Integrated Support System (ISS) Release 5 provides support software for a Wang computer system equipped with a Wang 2200VP or 2200MVP central processor. The major software components of ISS Release 5 are the following:

> ISS Utilities -- operator-controlled programs which provide various programming support and standard disk-related functions.
>
> Screen/Disk Subroutines -- program-controlled subroutines to simplify application programming by performing potentially complex operator- or disk-related functions.
>
> SORT-4 Subsystem -- a program-controlled subsystem which may be incorporated into an application program sort specified disk file records into a reordered output file according to the ascending or descending values of specified sort key fields. This program-controlled subsystem may be incorporated into an application program.
>
> Key File Access Method (KFAM) -- an indexed, disk file access method which supports multistation access to data file records according to ascending, descending, or random key sequence. Three versions of KFAM Release 7 are provided to accommodate different types of 2200 central processors and optionally multiplexed disk environments. KFAM Release 7 includes utilities, subroutines, and a subsystem.

Chapter 1 of this manual provides a software overview, lists the hardware requirements, describes ISS start-up procedures, and supplies other general information. Chapter 2 describes the ISS utilities. Chapter 3 describes the Screen/Disk Subroutines. Chapter 4 describes the SORT-4 Subsystem. Chapters 5 through 8 describe KFAM software. Five appendices are also included.

The following manuals contain information related to Wang 2200 hardware and should be available for reference purposes:

> The appropriate introductory manual supplied with the CPU.
>
> The Wang BASIC-2 Disk Reference Manual supplied with a disk or diskette drive (700-4081).
>
> The BASIC-2 Reference Manual supplied with the CPU (700-4080).

The appropriate disk user manual supplied with each disk drive.

If a printer is available, the appropriate printer manual supplied with the printer.

The ISS Release 5 software package consists of four prerecorded diskettes, a copy of the _ISS Release 5 Reference Card_, and a copy of this manual, the _Integrated Support System (ISS) Release 5 User Manual_. The ISS Release 5 package number is 195-0052-3. The diskette numbers are the following:

| Diskette Number | Diskette Name |
|---|---|
| 701-2423 (revision letter B or later) | ISS Utilities |
| 701-2424 (revision letter B or later) | Screen/Disk Subroutines |
| 701-2425 (revision letter B or later) | SORT-4 |
| 701-2427 (revision letter B or later) | KFAM-7 |

CONTENTS

CHAPTER 1 GENERAL INFORMATION

1.1 Software Overview . . . . . . . . . . . . . . . . . . . . . . 1
1.2 Hardware Requirements . . . . . . . . . . . . . . . . . . . . 7
1.3 Software Backup to Disk or Diskette . . . . . . . . . . . . . 9
1.4 ISS Start-up Instructions and Related Information . . . . . . . . . . . . . . . . . . . . . . 11
1.5 Application Program Requirements and ISS Common Variables . . . . . . . . . . . . . . . . . . . . 22

CHAPTER 2 THE ISS UTILITIES

2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . 24
2.2 The Copy/Verify Utility . . . . . . . . . . . . . . . . . . . 29
2.3 The Create Reference File Utility . . . . . . . . . . . . . . 31
2.4 The List/Cross-Reference Utility. . . . . . . . . . . . . . . 34
2.5 The Compression Utility . . . . . . . . . . . . . . . . . . . 36
2.6 The Decompression Utility . . . . . . . . . . . . . . . . . . 38
2.7 The Sort Disk Catalog Utility . . . . . . . . . . . . . . . . 39
2.8 The Disk Dump Utility . . . . . . . . . . . . . . . . . . . . 40
2.9 The File Status Report Utility. . . . . . . . . . . . . . . . 44
2.10 The Program Compare Utility . . . . . . . . . . . . . . . . . 45
2.11 The Reconstruct Disk Index Utility. . . . . . . . . . . . . . 46
2.12 The Alter Disk Index Utility . . . . . . . . . . . . . . . . 46
2.13 ISS Utilities Error Messages . . . . . . . . . . . . . . . . 50

CHAPTER 3 THE SCREEN/DISK SUBROUTINES

3.1 Introduction and Operating Notes . . . . . . . . . . . . . . 51
3.2 Data Entry (DEFFN'200) . . . . . . . . . . . . . . . . . . . 55
3.3 Date Routines (DEFFN'220,221,222,223,224,225) . . . . . . . . 59
3.4 Position Cursor (DEFFN'248) . . . . . . . . . . . . . . . . . 61
3.5 Operator Wait (DEFFN'254) . . . . . . . . . . . . . . . . . . 62
3.6 Re-enter (DEFFN'255) . . . . . . . . . . . . . . . . . . . . 62
3.7 Print Routine (DEFFN'242) . . . . . . . . . . . . . . . . . . 62
3.8 Select/Validate Disk Addresses (DEFFN'205) . . . . . . . . . 63
3.9 Search Index (DEFFN'229) . . . . . . . . . . . . . . . . . . 64
3.10 Allocate Data File Space (DEFFN'228) . . . . . . . . . . . . 64
3.11 Free Unused Sectors (DEFFN'227) . . . . . . . . . . . . . . . 65
3.12 Limits Next (DEFFN'226) . . . . . . . . . . . . . . . . . . . 66

3.13 Open/Close Output (DEFFN'240,241) . . . . . . . . . . . . . . 67
3.14 Open/Close Input (DEFFN'250,251) . . . . . . . . . . . . . . 69
3.15 Introduction to the Multiplexed/Multistation Disk Subroutines . . . . . . . . . . . . . . . . . . . . 70
3.16 Open (DEFFN'217) . . . . . . . . . . . . . . . . . . . . . . 72
3.17 End (DEFFN'218) . . . . . . . . . . . . . . . . . . . . . . 76
3.18 Close (DEFFN'219) . . . . . . . . . . . . . . . . . . . . . 77
3.19 Set/Release Hog Mode (DEFFN'215) . . . . . . . . . . . . . . 77
3.20 Translation Table Subroutines (DEFFN'201,202) . . . . . . . . 78

CHAPTER 4 THE SORT-4 DISK SORT SUBSYSTEM

4.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . 79
4.2 Writing the Setup Module . . . . . . . . . . . . . . . . . . 83
4.3 Input File Format Requirements . . . . . . . . . . . . . . . 88
4.4 Input Record Format Requirements. . . . . . . . . . . . . . . 90
4.5 Machine Configuration . . . . . . . . . . . . . . . . . . . . 103
4.6 Disk Device Addresses and Multistation Operation . . . . . . 105
4.7 Password Use . . . . . . . . . . . . . . . . . . . . . . . . 106
4.8 Sort Key Fields . . . . . . . . . . . . . . . . . . . . . . . 106
4.9 Sorting Partial Files . . . . . . . . . . . . . . . . . . . . 108
4.10 Type of Sort . . . . . . . . . . . . . . . . . . . . . . . . 109
4.11 Construction of Sort Records . . . . . . . . . . . . . . . . 112
4.12 The Sort Work File . . . . . . . . . . . . . . . . . . . . . 113
4.13 The Output File and Deferred Mounting . . . . . . . . . . . . 115
4.14 Special Input Record Selection Procedure . . . . . . . . . . 117
4.15 Exit from SORT-4 . . . . . . . . . . . . . . . . . . . . . . 121
4.16 Normal Operating Procedure . . . . . . . . . . . . . . . . . 122
4.17 Error Messages and Recovery Procedures . . . . . . . . . . . 123
4.18 SORT-4 Timings . . . . . . . . . . . . . . . . . . . . . . . 132
4.19 Sample Setup Programs . . . . . . . . . . . . . . . . . . . . 132

CHAPTER 5 KFAM-7 GENERAL INFORMATION

5.1 KFAM and Disk Access Methods . . . . . . . . . . . . . . . . 140
5.2 File Structures . . . . . . . . . . . . . . . . . . . . . . . 143
5.3 Software Components . . . . . . . . . . . . . . . . . . . . . 146
5.4 KFAM, the 2200 Device Table, and 2200 Disk Storage Characteristics . . . . . . . . . . . . . . . . . . . . . . 150
5.5 KFAM Record Types . . . . . . . . . . . . . . . . . . . . . . 154
5.6 General Requirements and Conventions . . . . . . . . . . . . 160
5.7 Choosing the Correct KFAM-7 Version . . . . . . . . . . . . . 165
5.8 Getting Started with KFAM-7 . . . . . . . . . . . . . . . . . 167

CHAPTER 6 THE KFAM-7 UTILITIES

6.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . 170
6.2 Initialize KFAM File . . . . . . . . . . . . . . . . . 176
6.3 Build Key File . . . . . . . . . . . . . . . . . . . . . 179
6.4 Print Key File . . . . . . . . . . . . . . . . . . . . . 180
6.5 Reallocate File Space . . . . . . . . . . . . . . . . . 183
6.6 Reorganize In Place . . . . . . . . . . . . . . . . . . 183
6.7 Reorganize/Rebuild Subsystem . . . . . . . . . . . . . 184
6.8 Convert to KFAM-7 . . . . . . . . . . . . . . . . . . . 191
6.9 Build Subroutine Module . . . . . . . . . . . . . . . . 191
6.10 Key File Recovery . . . . . . . . . . . . . . . . . . . 194
6.11 Reset Access Tables . . . . . . . . . . . . . . . . . . 195

CHAPTER 7 The KFAM-7 SUBROUTINES

7.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . 197
7.2 KFAM Access Modes . . . . . . . . . . . . . . . . . . . 202
7.3 General Programming Requirements . . . . . . . . . . . . 204
7.4 Guidelines for Multiple Key File and Duplicate Key Use . . 212
7.5 Open (DEFFN'230) . . . . . . . . . . . . . . . . . . . . 217
7.6 Delete (DEFFN'231) . . . . . . . . . . . . . . . . . . . 219
7.7 Findold (DEFFN'232) . . . . . . . . . . . . . . . . . . 220
7.8 Findnew (DEFFN'233) . . . . . . . . . . . . . . . . . . 221
7.9 Findnew(Here) (DEFFN'234) . . . . . . . . . . . . . . . 223
7.10 Findfirst (DEFFN'235) . . . . . . . . . . . . . . . . . 225
7.11 Findlast (DEFFN'236) . . . . . . . . . . . . . . . . . 225
7.12 Findnext (DEFFN'237) . . . . . . . . . . . . . . . . . 226
7.13 Findprevious (DEFFN'212) . . . . . . . . . . . . . . . 227
7.14 Release (DEFFN'238) . . . . . . . . . . . . . . . . . . 227
7.15 Re-open (DEFFN'213) . . . . . . . . . . . . . . . . . . 228
7.16 Write Recovery Information (DEFFN'214) . . . . . . . . . 228
7.17 Close (DEFFN'239) . . . . . . . . . . . . . . . . . . . 229
7.18 Non-KFAM Subroutines Included with KFAM-7 . . . . . . . . 230

CHAPTER 8 KFAM-7 TECHNICAL INFORMATION AND ADVANCED PROGRAMMING

8.1 Key File and Memory-Resident Control Information . . . . . 231
8.2 KFAM-7 File Names . . . . . . . . . . . . . . . . . . . 241
8.3 Findnew with Blocked Files . . . . . . . . . . . . . . . 242
8.4 Files Too Large for One Disk . . . . . . . . . . . . . . 243
8.5 Reusing Deleted Space with Findnew(Here) . . . . . . . . 244
8.6 Status of the Key Descriptor Record . . . . . . . . . . . 244
8.7 Key File Recovery Information . . . . . . . . . . . . . . 245
8.8 Key Index Records and the Adjustable Bias . . . . . . . . 245
8.9 Procedures for Changing User File Record Layout . . . . . 251

APPENDICES

Appendix A KFAM Utility Error Messages and Recovery Procedures . . . . . . . . . . . . . . . . . 253

Appendix B SORT-4 Variable Check-off List . . . . . . . . . . . 277

Appendix C Conditions Governing Spurious Results from the List/Cross-Reference Utility . . . . . . . 278

Appendix D Translation Tables: ASCII to EBCDIC and EBCDIC to ASCII . . . . . . . . . . . . . . . . . 280

Appendix E ISS Utilities Error Messages and Recovery Procedures . . . . . . . . . . . . . . . . . 282

Customer Comment Form . . . . . . . . . . . . . . . . . . . . . . . Last page

## FIGURES

Figure 1-1 ISS Release 5 Diskettes . . . . . . . . . . . . . . . . . . 1
Figure 1-2 Start-up Display with No Default Values . . . . . . . . . . 14
Figure 1-3 Start-up Display Format for Default Values. . . . . . . . . 16
Figure 1-4 The System Menu . . . . . . . . . . . . . . . . . . . . . . 17
Figure 1-5 The Application Menu Format . . . . . . . . . . . . . . . . 19
Figure 2-1 The ISS Utilities Menu. . . . . . . . . . . . . . . . . . . 25
Figure 2-2 Alter Disk Index Display Format . . . . . . . . . . . . . . 47
Figure 4-1 SORT-4 Sample Operation on Input Records . . . . . . . . . 111
Figure 4-2 Input Record Selection Flowchart. . . . . . . . . . . . . . 118
Figure 4-3 The SORT-4 Setup Program . . . . . . . . . . . . . . . . . 135
Figure 4-4 The INVTSORT Program . . . . . . . . . . . . . . . . . . . 138
Figure 4-5 The SORTINVT Program . . . . . . . . . . . . . . . . . . . 139
Figure 5-1 User File Structure . . . . . . . . . . . . . . . . . . . . 144
Figure 5-2 Key File Structure . . . . . . . . . . . . . . . . . . . . 145
Figure 5-3 Data Sector with Control Bytes . . . . . . . . . . . . . . 152
Figure 6-1 The KFAM-7 Utilities Menu . . . . . . . . . . . . . . . . . 171

## TABLES

Table 1-1 Minimum Size Requirements for the 2200MVP . . . . . . . . . 8
Table 1-2 ISS-5 Reference Files . . . . . . . . . . . . . . . . . . . 11
Table 1-3 Start-up and Menu Access Error Messages . . . . . . . . . . 20
Table 1-4 ISS Common Variables . . . . . . . . . . . . . . . . . . . 22
Table 2-1 ISS Utilities and Categories . . . . . . . . . . . . . . . 24
Table 2-2 Create Reference File Memory Requirements . . . . . . . . . 32
Table 2-3 Sort Disk Catalog Memory Requirements . . . . . . . . . . . 40
Table 2-4 Disk Dump SF Keys for CRT Output . . . . . . . . . . . . . 42
Table 2-5 Alter Disk Index SF Keys . . . . . . . . . . . . . . . . . 48
Table 3-1 Open Return Codes . . . . . . . . . . . . . . . . . . . . . 75
Table 4-1 SORT-4 Master Setup Program . . . . . . . . . . . . . . . . 84
Table 4-2 SORT-4 Input Record/File Format Combinations . . . . . . . 98
Table 4-3 Maximum Field Lengths and SORT-4 Field Lengths . . . . . . 101
Table 4-4 Alphabetic List of SORT-4 Error Messages . . . . . . . . . 124
Table 4-5 SORT-4 Timings, 2200VP . . . . . . . . . . . . . . . . . . 132
Table 4-6 Inventory Record Layout . . . . . . . . . . . . . . . . . . 136
Table 5-1 KFAM-7 Subroutines . . . . . . . . . . . . . . . . . . . . 148
Table 5-2 KFAM Record Types . . . . . . . . . . . . . . . . . . . . . 154
Table 5-3 Sector Layout for Record Type A Example . . . . . . . . . . 156
Table 5-4 Record Layout for Record Type M Example . . . . . . . . . . 160
Table 5-5 KFAM-7 Versions and Applicable Hardware . . . . . . . . . . 165
Table 6-1 KFAM Utility Use with Multiple Key Files . . . . . . . . . 173
Table 6-2 Information Required by Initialize KFAM File . . . . . . . 176

Table 6-3 SF Keys for Print Key File . . . . . . . . . . . . . . . . . 181
Table 6-4 Build Subroutine Module Options . . . . . . . . . . . . . . . 193
Table 7-1 KFAM Multistation Access Modes. . . . . . . . . . . . . . . 203
Table 7-2 KFAM Subroutine Argument Symbolic Variables . . . . . . . . 209
Table 7-3 KFAM Subroutine Q$ Return Codes . . . . . . . . . . . . . . 210
Table 8-1 The KDR Control Information . . . . . . . . . . . . . . . . 232
Table 8-2 KIR Control Information . . . . . . . . . . . . . . . . . . 234
Table 8-3 User Partition Storage of Array V0$() with the Non-Multiplexed KFAM-7 Versions . . . . . . . . . . . . 235
Table 8-4 User Partition Storage of Array T5$() with the Non-Multiplexed KFAM-7 Versions . . . . . . . . . . . . 235
Table 8-5 User Partition Storage of Array V0$() with the Multiplexed KFAM-7 Version . . . . . . . . . . . . . . 236
Table 8-6 User Partition Storage of Array T5$() with the Multiplexed KFAM-7 Version . . . . . . . . . . . . . . 237
Table 8-7 Table of Open Files, Array @T$(), per Entry . . . . . . . . 238
Table 8-8 Table of Protected Records, Array @V4$, per Entry . . . . . 240
Table 8-9 Description of the Files on the KFAM-7 Diskette . . . . . . 241
Table A-1 KFAM Error Message Categories and Recovery Options . . . . . . . . . . . . . . . . . . . . . . . . . . 254
Table A-2 KFAM Utility Error Messages . . . . . . . . . . . . . . . . 255
Table D-1 ASCII/EDCBIC Translation . . . . . . . . . . . . . . . . . . 280
Table E-1 ISS Utilities Error Messages and Recovery Procedures. . . . . . . . . . . . . . . . . . . . . . . . 282

# CHAPTER 1
# GENERAL INFORMATION

## 1.1 SOFTWARE OVERVIEW

The Integrated Support System (ISS) Release 5 fulfills a wide range of programming support and standard data processing needs for a disk- or diskette-based Wang computer system equipped with a 2200VP or 2200MVP central processor. The collection of software included in ISS Release 5 is supplied on four prerecorded diskettes. Each diskette contains a major component of ISS Release 5, as shown in Figure 1-1.

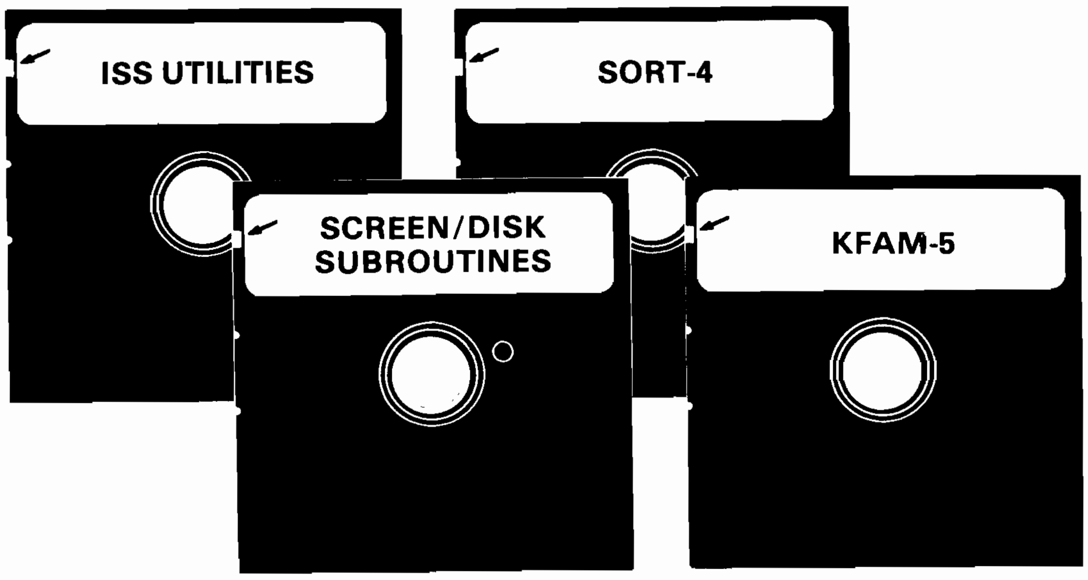


Figure 1-1. ISS Release 5 Diskettes

The major components of ISS are:

ISS UTILITIES -- A group of utility programs providing programming support and standard disk-related functions. The Utilities menu facilitates operator selection of the desired program.

SCREEN/DISK SUBROUTINES -- Designed for use with user-supplied application programs, these marked subroutines simplify operator/screen interaction, character code translation, and various disk-related functions. Subroutines are selected from a displayed list and are saved to disk for subsequent loading. Global subroutine use for 2200MVP systems is supported.

SORT-4 -- This versatile disk sort subsystem receives sort parameters from a short user-supplied program and copies records from the specified input file into an output file in reordered sequence. A variety of input record formats and file formats are supported.

KFAM-7 -- This disk file access method is comprised of subroutines and utilities. The subroutines support rapid access to randomly dispersed records according to ascending, descending, or random key sequence. KFAM is an acronym for Key File Access Method; the suffix "-7" denotes the Release 7 version of this support software. In ISS Release 5, KFAM-7 may be used with 2200VP or 2200MVP central processors and utilizes the global partition capabilities of a 2200MVP for common global storage of subroutines and record access control information.

ISS Release 5 also provides start-up software, which is contained on the ISS Utilities, Screen/Disk Subroutines, and KFAM-7 diskettes.

ISS software may be copied to a single fixed/removable disk for efficient program loading and multiuser access via a multiplexed disk drive.

## ISS Start-up

Each 2200VP central processor or 2200MVP memory partition is considered by ISS Release 5 (ISS-5) conventions to be a "station." Each station begins ISS start-up via a program file called START. During start-up, the user defines the current date and available peripherals. This information is made available to all ISS software and application programs running in that station, along with certain information defined internally by ISS start-up software. ISS start-up typically occurs at the beginning of a working day and also whenever peripheral availability changes.

ISS start-up software supplies access to all menu-controlled ISS components through its flexible menu structure and allows the user to specify the desired menu after start-up has been completed. In addition, a user-specified application program may be loaded and run by specifying the Application menu.

## ISS Utilities

The ISS utilities are operator-controlled programs that provide various programming- or disk-related functions. Several utilities support several forms of multiple file processing; all utilities are compatible with multiplexed/multistation operation.

COPY/VERIFY -- copies one or more files from one disk to the same or a different disk and optionally verifies the copy. Each file copied to the output disk may be either a new file or a replacement for an existing file. The number of extra sectors may be specified to determine the allocated space for the output file.

CREATE REFERENCE FILE -- allows the user to create, edit, or print a "reference file." A reference file consists of multiple entries in the form of an input file name, an output (or second) file name, and an extra sectors value. Many ISS utilities allow use of a reference file to specify the files to be processed.

LIST/CROSS-REFERENCE -- lists and/or cross-references each program file specified. If cross-referenced, printed tables indicate: (1) for each line number referenced elsewhere in the program, a list of line numbers referencing that line number, (2) for each variable used, a list of line numbers in which that variable appears, (3) a list of each marked (DEFFN' statement) subroutine's line number followed by a list of line numbers which reference that marked subroutine.

COMPRESSION -- copies each specified program file to an output file in compressed form, which eliminates spaces, inessential line numbers, and REM statement lines. The compressed program requires less memory and executes faster than the original program.

DECOMPRESSION -- copies each specified program file to an output file in decompressed form, separating each multistatement line into single statement lines and assigning a unique line number to each executable BASIC-2 statement.

SORT DISK CATALOG -- prints or displays a report of a disk catalog index. File names may be sorted: (1) alphabetically by file name, (2) by ascending starting sector number, or (3) by ascending file sequence in the disk index.

DISK DUMP -- allows the user to display or print the contents of one or more contiguous sectors within a specified file or range of absolute sector numbers. Each byte's hexadecimal value and corresponding characters are displayed in one format or can be printed in one of three formats, including printing records formatted with control bytes (e.g., SOV, EOB) on a field-by-field basis.

FILE STATUS REPORT -- performs several functions pertaining to multiplexed/multistation data files, including closing the specified files for one or all possible stations and printing or displaying the status of specified files relative to either all possible stations or a specified station.

PROGRAM COMPARE -- for each pair of program files specified, a comparison is made on a line-number-by-line-number basis. Messages appear indicating statement lines whose contents do not match, if a statement line number exists in only one of the program files, if one program file ends while the other continues, and if all pairs of program files are identical.

RECONSTRUCT DISK INDEX -- reconstructs a disk catalog index in the event of its accidental destruction; e.g., a scratched disk.

ALTER DISK INDEX -- performs certain functions involving a disk's catalog index, which are: displaying the contents of the catalog index, renaming a file, changing an active (unscratched) file to a scratched file, changing a scratched file to an active file, searching for a specified file name and displaying its disk sector allocation and use, and removing the last file allocated on a disk thereby freeing its disk space for allocation to other files.

## Screen/Disk Subroutines

There are three groups of Screen/Disk subroutines: Screen subroutines, Disk subroutines, and Translation Table subroutines. After choosing the desired subroutines, the user specifies whether the chosen subroutines are to be used as nonglobal or global subroutines. With nonglobal use, both variables and subroutines are incorporated into the user's application program. With global use, only the variables are incorporated into the application program, and the subroutines reside in a global partition accessible to multiple partitions.

### Screen Subroutines:

DATA ENTRY -- displays a prompt and accepts a numeric or alphanumeric keyboard entry. Checks are made to determine whether the entry falls within specified alphanumeric limits or a specified numeric range, whether the length is acceptable, whether the entry is on a list of valid entries (table look-up), and whether a numeric entry has an acceptable number of digits to the left and right of the decimal point. If unacceptable, entry is rerequested. A displayed operator-modifiable default value may be implemented.

POSITION CURSOR -- positions the cursor to a specified location on the CRT screen and optionally erases characters to the right of the cursor's position on the same line and lines below the cursor's position.

DATE ROUTINES -- allows entry of dates in Gregorian or Julian form and allows conversion between Gregorian and Julian dates. The difference between two dates may also be calculated.

OPERATOR WAIT -- displays KEY RETURN(EXEC) TO RESUME and waits for the RETURN key to be touched.

PRINT -- allows a specified character to be printed a specified number of times.

RE-ENTER -- displays RE-ENTER to indicate an invalid operator entry.

Disk Subroutines:

SELECT/VALIDATE DISK ADDRESSES -- validates a specified disk address by checking that it is a valid 2200 disk address and optionally selects the disk address to a specified file number using a SELECT statement.

SEARCH INDEX -- examines a disk catalog index to determine if a specified file is: (1) cataloged, (2) scratched or active, and (3) a data or program file.

ALLOCATE DATA FILE SPACE -- opens a new file on the specified disk and allocates to that file either a specified number of sectors or all sectors beyond the current end of allocated disk space to the end of the catalog area.

FREE UNUSED SECTORS -- examines a specified file's END (end-of-data) control sector and the catalog trailer (end-of-file) control sector and reallocates the unused sectors (between the END control sector and catalog trailer control sector) as "free" sectors available for other files. This reallocation reduces the number of extra sectors within the file to zero; however, a minimum number of extra sectors may be specified to limit the number of sectors reallocated as free disk space.

OPEN/CLOSE OUTPUT -- opens for output or closes a data file containing certain special-purpose software header and trailer records.

OPEN/CLOSE INPUT -- opens for input or closes a data file containing certain special-purpose software header and trailer records.

LIMITS NEXT -- returns the name of the next file according to file sequence in a specified disk's index and indicates whether the file is scratched or active and a data or program file.

MULTIPLEXED/MULTISTATION FILE OPEN/END/CLOSE -- These subroutines control multiple station access to specified data files. When opening a file, one of four access modes is specified. Whether or not file access is granted depends upon the file access mode specified and the file access mode already granted to other stations for that file. These subroutines support opening a new file, opening an existing file, writing an END trailer record (equivalent to DATASAVE DC END), and closing a file. File password protection is supported, and a disk hog mode option is available.

Translation Table Subroutines:

ASCII to EBCIDIC -- assigns the proper hexadecimal codes to an alphanumeric array (table) for the code translation from ASCII to EBCIDIC.

EBCIDIC to ASCII -- assigns the proper hexadecimal codes to an alphanumeric array (table) for the code transalation from EBCIDIC to ASCII.

## SORT-4 Disk Sort Subsystem

SORT-4 is a subsystem designed to sort the records in a specified input file into a reordered sequence in a specified output file. Up to ten ascending or descending fields in each record collectively compose the sort key, which determines output file record order. Three types of sorts are available: (1) a full-record sort, (2) a key sort, and (3) a tag sort which outputs only sorted pointers to the records in the input file. Acceptable input file formats include general sequential files; BAS-1 sequential files; ISS Open/Close sequential files; and KFAM-3, KFAM-4, KFAM-5, and KFAM-7 files. Files containing variable length records, including 2200 TC (Telecommunications) formatted files, are also supported by SORT-4. Record formats include unpacked records, packed subfields, and packed records written by DATASAVE DC, DATASAVE DA, or DATASAVE BA statements (with certain exceptions). Input record selection to include or exclude records based on logical tests performed by user-supplied program statements is also supported.

A short, user-supplied setup program loads SORT-4 software and supplies sort parameters such as input file name, input file format, input record format, and related information.

## KFAM-7

This software system is designed to create, maintain, and search an index to the records in a data file. All records within the data file must be a fixed length and contain an alphanumeric key located within the same portion of every record; the key must be unique when compared to the keys contained within other records in the file, unless special considerations and procedures are observed which allow duplicate key use. The index, contained within a separate Key File, contains a key for each record and the location of that record. Multiple Key Files for a single data file allow access to records in the data file using more than one key field. Control information supporting multistation record access and record protection using one of four available access modes is also contained in (1) the Key File with the "Multiplexed" KFAM-7 version, (2) in a global partition with the "Single Bank" KFAM-7 version, or (3) in a universal global partition with the "Multiple Bank" KFAM-7 version.

KFAM-7 subroutines either reside as global program text in a 2200MVP partition or are incorporated into a user-supplied application program to perform routine functions on the index, such as random key search, ascending or descending key sequence search, adding and deleting key/record entries.

KFAM-7 utilities can create a KFAM-7 Key File based upon a sequential data file containing fixed-length, fixed-format records, create the index in the Key File, reorder data file records and Key File entries into ascending key sequence, print certain Key File information, and perform other functions described elsewhere in this manual.

KFAM-7 utilizes the 2200MVP memory partitioning scheme, allowing KFAM-7 subroutines and record access/protection control information to be contained within a global partition accessible to application programs and KFAM-7 utilities running in other partitions. The several versions of KFAM-7 are made available by the presence of different global program files. Programming differences between the several KFAM-7 versions are nearly transparent to user software.

## 1.2 HARDWARE REQUIREMENTS

ISS-5 requires either a 2200VP or a 2200MVP central processor. With a 2200VP, a minimum of 16K memory and Release 1.9 (or a subsequent release) of the 2200VP Operating System are required. With a 2200MVP, the minimum partition size required depends upon the ISS component to be used (as described in Table 1-1), and Release 1.7 (or a subsequent release) of the 2200MVP Operating System is required.

Peripheral requirements include a disk or diskette drive and, for certain ISS software, a printer. Either one Model 2270, 2270A, or 2270AD Diskette Drive must be available for mounting an ISS diskette or one fixed/removable disk must contain the ISS-5 software. At least one other fixed/removable disk or a second diskette drive (in a dual or triple drive) should be available for data file storage, especially if the ISS-5 software is to reside on diskettes. Supported disk device addresses are 310/B10, 320/B20, 330/B30, 350/B50, 360/B60, 370/B70 and Model 2280 disk device addresses D10 through D15, D20 through D25, D30 through D35, D50 through D55, D60 through D65, and D70 through D75. It is recommended that each disk or diskette drive be labeled with its respective disk device address.

A Wang printer (132-column) is recommended for all ISS-5 software and is required for the List/Cross-Reference ISS utility. Supported printer addresses are 204, 211, 212, 213, 214, 215, 216, and 217. In addition, an output address of blank indicates user-interactive CRT screen output. An output address of 01D indicates output via a telecommunications interface controller.

In a system where a disk multiplex controller is used, an Engineering Change (ECN) to the multiplex controller may be necessary in order to use the disk Multiplexed version of KFAM-7.

Table 1-1. Minimum Size Requirements for the 2200MVP

| ISS COMPONENT | PARTITION SIZE REQUIREMENTS |
|---|---|
| ISS Start-up Module | 8.5K |
| ISS Utilities | Copy/Verify - 10K<br>Create Reference File - 8.5K minimum (See Chapter 2)<br>Compression - 13.5K<br>Decompression - 11.5K<br>Sort Disk Catalog - 8.5K minimum (See Chapter 2)<br>Disk Dump - 9K<br>File Status Report - 10.5K<br>Program Compare - 14K<br>Reconstruct Disk Index - 8.5K<br>Alter Disk Index - 11.25K |
| Screen/Disk Subroutines | To choose the desired subroutine - 12K<br>All subroutines are chosen - 10K (8.9K memory use)<br>Variables necessary for global use - 4K (2.8K memory use) |
| SORT-4 Subsystem | Sequential file - 9K<br>KFAM file - 11-12K<br><br>SORT-4 adjusts itself to the amount of memory available and provides better throughput if run in a larger size partition. |
| KFAM-7 | If global KFAM subroutines are not available, program overlays are loaded, usually increasing partition size requirements.<br><br>Initialize KFAM File - 10.25K<br>Build Key File - 9.75K (13.25K with overlays)<br>Print Key File - 9.25K (12.75K with overlays)<br>Reallocate File Space - 9K (12.75K with overlays)<br>Reorganize in Place - 9.5K (13.25K with overlays)<br>Reorganize/Rebuild Subsystem - 9.25K (13K with overlays)<br>Convert to KFAM-7 - 9.25K (10.25K with overlays)<br>Build Subroutine Module - 7.75K<br>Key File Recovery - 9.75K (11K with overlays)<br>Reset Access Tables - 9K |

Table 1-1. Minimum Size Requirements for the 2200MVP (continued)

| ISS COMPONENT | PARTITION SIZE REQUIREMENTS |
|---|---|
| KFAM-7 User Variables | Approximately 1000 bytes, plus 87 bytes per accessed KFAM file. |
| KFAM-7 Global Subroutines and Global Variables | Multiplexed version - 8K per bank<br>Single Bank version - 9.75K per bank<br>Multiple Bank version:<br>global subroutines - 8.5K per bank<br>global variables - 2.75K in the universal global area. |

## 1.3 SOFTWARE BACKUP TO DISK OR DISKETTE

ISS-5 software is supplied on four diskettes. A copy should be made of each diskette. Once copied, the supplied ISS-5 diskettes should be kept as backup software, and the newly copied diskettes should be labeled and used. If preferred, one or more ISS-5 diskettes may be copied to a single fixed/removable disk.

### Copy Guidelines

To make diskette backup copies if a dual (or triple) diskette drive is available, use the COPY statement followed by the VERIFY statement.

> NOTE:
>
> If the input diskette was formatted using a 2200C, 2200S, or 2200T central processor, use the Copy/Verify utility instead of these procedures.

For example, to copy an ISS-5 diskette in the leftmost diskette slot at disk device address xyy to a formatted diskette which does not contain irreplaceable data in the rightmost diskette drive (dual drive) or middle diskette drive (triple drive), enter the following sequence:

```
:SELECT DISK xyy
:COPY FR (0,1023)
:VERIFY R (0,1023)
```

To copy one or more ISS-5 diskettes to a single fixed/removable disk, proceed as follows:

1. Obtain a fixed/removable disk which does not contain irreplaceable data or program files. The disk must have been formatted at one time and must be scratched using the SCRATCH DISK statement immediately prior to copy.

2. Mount the ISS-5 disk(ette) containing the ISS utilities. Load ISS start-up software and complete the start-up procedures as described in Section 1.4.

3. From the ISS Utilities menu, choose the Copy/Verify utility. Set Copy/Verify parameters as follows:

   INPUT ADDRESS - xyy (diskette address)
   INPUT OPTION - COPY/VERIFY
   MODE - ALL
   OUTPUT ADDRESS - xyy (fixed/removable disk address)
   OUTPUT OPTION - ADD
   EXTRA SECTORS - -1

4. Refer to Chapter 2 for information on operating the Copy/Verify utility. Mount the disk and the ISS diskette at their specified addresses. If error messages of the following type appear: FILE - filename - CANNOT BE COPIED, skip to Step 7. If error messages do not appear, upon completion of copy refer to Step 5.

5. Remove the diskette just copied and replace it with the next diskette to be copied; refer to Step 3 to copy the next diskette. After copying all diskettes, refer to Step 6.

6. After copying the appropriate ISS diskette, touch SF'31 as required to obtain the start-up display shown in Figure 1-3. Change the ISS loading address to the address of the fixed/removable disk.

7. Obtain the ISS Utilities menu and choose the Copy/Verify utility. Set Copy/Verify parameters identical to those in Step 3, except set the Input Mode to Verify.

8. After completion of verification, if an error message did not appear, remove the diskette just verified and replace it with the next diskette to be copied; refer to Step 3 to copy the next diskette. Otherwise, see Appendix E.

## <u>ISS-5 Reference Files</u>

ISS-5 supplies reference files primarily for use with the ISS utility Copy/Verify, MODE = INDIRECT. For example, if all ISS diskettes have been copied to a single fixed/removable disk, other diskette copies of the four ISS components may be made using the appropriate reference file names. The reference file names and associated ISS components appear in Table 1-2.

Table 1-2. ISS-5 Reference Files

| REFERENCE FILE NAME | ISS COMPONENT |
|---|---|
| ISS.REF1 | The ISS Utilities diskette. |
| ISS.REF2 | The ISS Screen/Disk Subroutines diskette. |
| KFAMREF7 | The KFAM-7 diskette. |
| SORTREF4 | The SORT-4 diskette. |
| KFAMREFS | KFAM-7 Reorganize/Rebuild Subsystem. |

## 1.4 ISS START-UP INSTRUCTIONS AND RELATED INFORMATION

ISS start-up operation allows the user to enter, in reply to operator prompts, certain information pertaining to the station currently in use. Following ISS start-up operation, the information pertaining to that station, including available peripheral addresses and the date, is available (via common variables) to all software running in that station. 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 printouts.

Both operator-entered information and internally obtained information tested during start-up is available to all software running in a station. The operator entered and internally obtained information is collectively referred to as the "system configuration table" for the station in use. Internally obtained information includes memory size and type of CPU.

Operator-entered start-up information is automatically saved (recorded) in a station file on the ISS disk(ette). During subsequent ISS start-up operations, the current contents of the station file are automatically loaded and appear as operator-modifiable default values. A station file may be created during start-up operation for a particular station number and occupies 10 disk sectors. Station file names are assigned as ISS.0nnD, where nn is the station number from 1 through 48. Default values for start-up operation and most ISS utilities are maintained within this station file.

ISS start-up operation typically occurs only at the start of the working day, following Master Initialization of the central processor. During the day, however, if Master Initialization (partition generation with a 2200MVP central processor) is repeated or the user needs to change any system configuration table values, ISS start-up operation must be repeated.

## Recommended Station Number Convention

The first prompt which appears during ISS start-up operation requests entry of a station number. The station numbers in use at any time must be unique, or potentially dangerous file access problems may occur. One way to ensure unique station numbers are always in use with a 2200MVP central processor is to assign each station number as the partition number currently in use. A unique station number must be assigned to each station accessing shared disk(ette) drives because the entered station number (any number from 1 through 48) is equated to common variable S2, and is used by utilities (and possibly application software) to determine which station is accessing a multiplexed/multistation disk file.

> NOTE:
>
> Although ISS start-up allows station numbers from 1 through 48, the following ISS components only allow station numbers from 1 through 16: KFAM-7 utilities and subroutines, and the SORT-4 subsystem when sorting a KFAM file. It is recommended that these station number limitations be considered before assigning unique station numbers.

## 2200MVP Partition Generation Considerations

As discussed in the 2200MVP Introductory Manual, the 2200MVP system disk contains the 2200MVP Operating System and other software associated with partition generation. Following Master Initialization, the partition generation phase allows 2200MVP memory to be allocated to the partitions to be used, and each terminal may be assigned to one or more partitions. Partition size requirements for the ISS components appear in Table 1-1.

During partition generation, if the peripheral addresses have not yet been defined in the 2200MVP's Master Device Table, choose the Edit Device Table option. Any peripheral device address used during ISS operation must have previously been entered into the Master Device Table. The Master Device Table's current (memory-resident) values may become its default (disk-resident) values by saving a partition configuration (Save Partition Configuration option) before executing the partition configuration. Thereafter, the Edit Device Table option is needed only if the default values (stored on the 2200MVP system disk) are not acceptable.

If global program files are to be used, it is recommended that these program files be copied to the 2200MVP system disk to allow the automatic bootstrap feature to be implemented. The ISS Copy/Verify utility may be used to copy such files following ISS start-up.

After executing a partition configuration, the partition number appears on the screen. The $RELEASE TERMINAL statement may be used to change the partition currently attached to the terminal (e.g., if the terminal in use has been assigned more than one partition).

## Edit Mode and Use of Special Function Keys

The top row of keys on the operator's keyboard are referred to as Special Function Keys. When the SHIFT key is not depressed (unshifted position), Special Function Keys '0 through '15 are available; when the SHIFT key is depressed (shifted position, the SHIFT light illuminates), Special Function Keys '16 through '31 are available.

If the operating instructions below request the operator to touch any Special Function (SF) Key, Edit mode must not be active. With a 2200VP, Edit mode is indicated by an asterisk (*) to the left of the entry; with a 2200MVP, Edit mode is indicated by a blinking cursor. Edit mode, if active, may be switched off by manually touching the EDIT key once, which erases the asterisk or causes the cursor to stop blinking. Similarly, Edit mode may be activated by touching the EDIT key again.

## Valid Keys for Editing Entry Fields

ISS-5 provides a convention whereby certain keys are available for destructive or nondestructive editing of displayed defaults or entries for all ISS-5 prompts.

The BACKSPACE and LINE ERASE keys are always available for destructive editing. BACKSPACE moves the cursor one position to the left and places a blank at the new cursor position; LINE ERASE erases all characters in an entry field.

When Edit mode is active, certain SF keys become Edit keys and are available for destructive editing (ERASE and DELETE keys) and nondestructive editing (BEGIN, END, INSERT, →, ——→, ←——, ←). Refer to the BASIC-2 Reference Manual for a description of these Edit keys.

For fields requesting a numeric value, leading zeros need not be entered.

## Loading ISS Start-up Software

If an ISS diskette is to be used, it must contain the desired ISS software component, and a tab must be in place over the write protect hole. Please note that the word "enter," as used in the following instructions, indicates that the appropriate characters are to be entered and visually verified, and then the RETURN key should be touched to complete the entry. Mount the ISS disk(ette) and enter the following sequence:

```
SELECT DISK xyy

LOAD RUN
```

(where xyy is replaced by the disk device address where the ISS disk(ette) has been mounted)

## Responding to the Start-up Prompts

After loading start-up software, the prompt ENTER STATION NUMBER appears in the upper-left corner of the screen. There are three options available when responding to this prompt:

1. To view existing station file numbers and, optionally, the start-up default values contained within specified station files, touch SF'0. The station numbers whose station files are found on the ISS disk(ette) are displayed. For each specified station number, the corresponding default values appear. To redisplay the ENTER STATION NUMBER prompt, enter 0 (zero).

2. To create a station file for a particular station number, touch SF'16. In reply to the ENTER STATION NUMBER TO CREATE prompt, enter the desired station number. If the station number is acceptable, the station file is created with all default values set to blanks; the ENTER STATION NUMBER prompt reappears. Otherwise, entry is rerequested. The station number must be between 1 and 48, there must be sufficient disk space to create the file, and a station file (active or scratched) must not already exist on the ISS disk(ette) for that station number.

3. To proceed without viewing the existing station files or creating a new station file, enter a station number from 1 through 48. If the entry is unacceptable, entry is rerequested; this indicates that the station file for this station number either does not exist or has been scratched (see Option 2 above). If the station file is located, the next sequence of prompts appear and the operator either enters all system configuration table default values if the station file contains no default values, or views and optionally modifies the displayed default values contained within the station file.

## Case 1: No Default Values in the Displayed Station File

If the station file contains no default values, the screen appears as shown in Figure 1-2.

```
ENTER TODAY'S DATE        (MM/DD/YY)
?_

    STATION #nn

1   DATE                  -
2   MENU TO LOAD          -
3   PRINTER ADDRESS       -
4   LOADING ADDRESS       -
5   DISK ADDRESSES        -
```

Figure 1-2. Start-up Display With No Default Values

A reply to each item is requested beginning with DATE. The station number is displayed (nn in Figure 1-2).

The current date is entered in the form mm/dd/yy (month/day/year). After entry, the date is displayed and the ENTER PRINTER ADDRESS prompt appears.

The MENU TO LOAD prompt is accompanied by a list of available menus. The Utilities (ISS utilities), Screen/Disk, and KFAM-7 menus allow selection of the individual options available for each of these software components. The SYSTEM menu is a higher-level menu which is especially convenient for ISS software residing on a fixed/removable disk; it allows selection of the Utility, Screen/Disk, or KFAM-7 menu. The Application menu allows a specified application program to be automatically loaded and run. The Application menu, System menu, or the menu(s) corresponding to the ISS software component contained on the currently mounted ISS disk(ette) may be specified.

Enter the number (1 through 6) which corresponds to the menu to be loaded. The name of the menu is displayed.

The PRINTER ADDRESS prompt is accompanied by a list of supported printer addresses. The entered address determines the output device used during utility program operation for most functions. If a printer is not available, touch the RETURN key without entering any characters (blank); user-interactive output appears on the screen when the printer address is blank, accompanied by the prompt KEY RETURN (EXEC) TO RESUME and requires that the RETURN key be touched for the program to continue. An output address of 000 indicates no output, where error messages encountered during utility operation are not viewable. An output address of 005 indicates CRT output where the CRT is treated like a printer, where error messages appear so briefly on the CRT that they may not be viewed. Output addresses of 000 and 005 should be used with caution and only in situations where error messages are irrelevant. An output address of 01D indicates output via a Wang Telecommunications Controller. If a printer connected to an operator's 2200MVP terminal (called a local printer) is to be used, enter 204. Otherwise, enter the three-digit printer address. The printer whose address is entered should be powered ON and SELECTed. Any nonblank entry is displayed. It is recommended that any printer whose address is entered be visually checked to ensure that it is switched on and selected, especially if a local 2200MVP terminal printer or a Wang Printer Multiplexer is to be used. Note that ISS and KFAM utilities hog the printer device when appropriate.

The LOADING ADDRESS prompt is the disk address from which ISS software is to be loaded. This entry may be one of the disk addresses already entered. In most cases, this is the disk address from which ISS software was just loaded.

The DISK ADDRESS prompt, ENTER DESIRED OPTION (0=END), appears along with 1-ADD, 2-CHANGE ALL, 3-DELETE, where each option is chosen by entering its corresponding number (1, 2, or 3). The ADD option allows each disk address entered to be added to the disk addresses previously displayed. The CHANGE ALL option erases any previously displayed disk addresses and allows entry of the desired disk addresses. The DELETE option allows one or more of the displayed disk addresses to be entered and removed from the list. The ADD and CHANGE ALL options, when chosen, display all ISS-supported disk addresses. After choosing any of the options, enter a 0 (zero) to indicate completion, which, if entered before any disk addresses are entered, allows the chosen option to be aborted.

Only the disk addresses and the ISS loading address may be specified during utility operation. The disk addresses may not include the disk address specified for the loading address. If the currently displayed loading address is incorrect, first change the loading address and then add the old loading address to the list of disk addresses, if desired. After modifying the list of disk addresses, enter 0 (zero) to obtain the next prompt.

The screen now displays the entered default values. Refer to "Case 2: Default Values in the Displayed Station File."

## Case 2: Default Values in the Displayed Station File

If default values have been loaded from the station file or manually entered from the keyboard, these values are displayed in the formats shown in Figure 1-3.

```
ENTER DESIRED FUNCTION (0=END)

_

     STATION #nn

1    DATE                   - mm/dd/yy
2    MENU TO LOAD           - 11111
3    PRINTER ADDRESS        - abb
4    LOADING ADDRESS        - xyy
5    DISK ADDRESS           - xyy xyy xyy
```

Figure 1-3. Start-up Display Format for Default Values

The displayed default values should be visually verified by the operator. To change one of the values, enter the number (1 through 5) of the value to be changed. For additional instructions, refer to the appropriate paragraph under "Case 1: No Default Values in the Display Station File." The ENTER DESIRED FUNCTION (0=END) prompt reappears after entering a valid response to any default value. If the station number (STATION #nn) previously entered is incorrect, touch SF'31 to return the ENTER STATION NUMBER prompt to the screen.

When the displayed information is correct, enter 0 (zero). The displayed default values are saved in the station file and become the default values for subsequent start-up operations. The menu indicated as the MENU TO LOAD soon appears on the screen. Should an error message appear, see Table 1-3.

## ISS Menu Hierarchy and SF'31

The highest level of the three-level ISS menu hierarchy is start-up operation. Before any ISS menu can be obtained, ISS start-up operation must first occur. One step below start-up is the System menu. Below the System menu each of the following menus has third-level status: Utilities, Screen/Disk Subroutines (three menus), KFAM-7, and Application. SF'31 may be touched in reply to any start-up prompt, any of the ISS menus, or any utility program prompt to abort the current operation and obtain the next-highest step in the ISS menu hierarchy. For instance, if operating an ISS utility program, touching SF'31 returns the Utilities menu to the screen. If SF'31 is touched again, the Utilities menu is replaced by the System menu and, if touched again, the System menu is replaced by the start-up default values.

Except for the Application menu, each menu lists the options available, and an SF key number appears to the left of each option. To select an option, touch the corresponding SF key.

Figure 1-4 shows the System menu from which the listed ISS components may be chosen by touching the corresponding SF key. For example, to load (obtain) the Application menu, touch SF'03.

```
SELECT SUB-MENU

                    SYSTEM MENU (STATION # = n)
-----------------------------------------------------------------
  FN KEY        PROGRAM NAME              FN KEY      PROGRAM NAME
-----------------------------------------------------------------
    01     ISS UTILITIES                    03     APPLICATIONS

    02     SCREEN/DISK SUBROUTINES          04     KFAM-7

                                            31     RE-START SYSTEM
```

Figure 1-4. The System Menu

CAUTION:

Following ISS start-up, never remove a data disk or diskette which was accessed by the program in use unless an ISS menu appears or a utility prompt requests that a disk(ette) be mounted. Neither touch the RESET key, nor switch the CPU's (or terminal's) power off unless one of the ISS menus appears on the screen. Failure to obey these procedures can result in the accidental destruction of data.

It is recommended that the System menu appear on the screen if an ISS disk(ette) is to be replaced by a different ISS disk(ette). Station file default values are automatically updated and a station file is automatically created (if one does not exist for this station number) when a different ISS disk(ette) is mounted at the ISS loading address while the System menu or the error message NOT AVAILABLE ON CURRENT DISK appears.

In order to obtain an ISS menu instead of responding to a utility program prompt, touch SF'31. By touching SF'31, any data file currently open will be closed. Data files accidentally left open may cause file access conflicts.

## Loading an Application Program

If APPLICATION is the specified MENU TO LOAD during the start-up phase or APPLICATION is chosen in reply to the System menu, the Application menu appears on the screen. In this menu, the default values may be displayed with the prompt ENTER OPTION TO CHANGE (0=LOAD APPLICATION) as shown in Figure 1-5.

However, if default values do not exist, the Application menu appears with the prompt ENTER APPLICATION TO LOAD. After entering the file name of the application program to be loaded and run, the prompt ENTER ADDRESS TO LOAD FROM appears. Valid disk device addresses from which an application program may be loaded include all ISS start-up disk addresses and the ISS loading address. After entering the three-digit disk address from which the program file is to be loaded, the Application menu appears as if the station file contained the default values (see Figure 1-5).

```
ENTER OPTION TO CHANGE (0 = LOAD APPLICATION)

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

STATION NUMBER  - n                  CPU TYPE        - 11
GREGORIAN DATE  - mm/dd/yy           JULIAN  DATE    - nnnnn
MEMORY SIZE     - nn    K            CRT WIDTH       - nn
LOADING ADDRESS - xyy                PRINTER ADDRESS - abb
DISK ADDRESSES  - xxy xyy
```

Figure 1-5. The Application Menu Format

The Application menu displays the common variable values present in this station's system configuration table. If the name appearing to the right of APPLICATION TO LOAD is incorrect, enter 1; then enter the file name of the application program to be loaded. If the disk address appearing to the right of APPLICATION DISK ADDRESS is incorrect, enter 2; then enter the disk address from which the specified application program is to be loaded.

When the displayed values for items 1 and 2 are correct, enter 0 (zero) to load and run the specified application program. If an error message appears, refer to Table 1-3.

## <u>ISS Start-up and Menu Access Error Messages</u>

ISS provides certain error messages to facilitate recovery from typical operational errors which may occur during the start-up and menu access phase. Table 1-3 lists each error message and its associated recovery procedures. Also included are some typical "hardware" errors (ERR 1nn form) which may result from incorrect operation.

Table 1-3. Start-up and Menu Access Error Messages

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| ERR A01 | Insufficient memory available to perform the specified operation. | If a 2200MVP is in use, see Section 1.2 for the the partition size needed for this operation. Touch SF'31. Either use a station with a larger memory size or (2200MVP only) wait until a partition configuration is re-executed to increase this partition's memory size. |
| ERR I95 | The write protect hole is not covered when attempting to write data (e.g., default values) on the specified diskette. This error could also be caused by a disk hardware (seek) error. | Touch SF'31. Remove the diskette and place a tab over its write protect hole; remount the diskette and retry. If the tab is in place, make sure the diskette spins freely in its jacket; remount it and retry. If unsuccessful, retry using a different disk(ette). If this error persists, contact a Wang Service Representative. |
| ERR I98 | A disk(ette) is not mounted (or READY), or the diskette drive door is open. | Touch SF'31. Check if the disk(ette) is mounted. Retry, specifying the correct disk address. |
| ERR P48 | A peripheral device address was specified which does not exist in this 2200MVP's Master Device Table. | Touch SF'31. Do not use the specified address until it has been entered into the Master Device Table; i.e., during partition generation. |
| ERR lnn | For all other ERR lnn error codes, refer to the <u>Wang BASIC-2 Language Disk Reference Manual</u> for disk-related errors or the <u>BASIC-2 Reference Manual</u> for other errors. | Touch SF'31. |

Table 1-3. Start-up and Menu Access Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| FILE filename IS NOT AT ADDRESS xyy KEY RETURN(EXEC) TO RESUME? | When attempting to load an application program, the specified program file (filename) does not exist at the specified disk address (xyy). | Either touch SF'31 to facilitate re-entry of the correct application program file name and disk address, or mount the correct disk and touch the RETURN key. |
| LOADING ADDRESS WRONG. KEY RETURN(EXEC) TO RESUME? | The disk(ette) at the specified (start-up) ISS loading address either does not contain ISS software or is not mounted there. | Touch the RETURN key. Either change the ISS loading address, or mount a disk(ette) containing ISS software and continue. |
| MVP OPERATING SYSTEM RELEASE 1.7 (MINIMUM) REQUIRED, KEY RETURN (EXEC) TO RESUME? | ISS-5 requires that the 2200MVP Operating System (OS) be Release 1.7 or more recent. | When all other stations have completed their respective applications, from terminal 1 use an Immediate mode $INIT statement to allow Release 1.7 (or more recent) of the 2200MVP OS to be loaded if available. If not available, obtain Release 1.7 (or a more recent release) from Wang Laboratories, Inc. |
| NOT AVAILABLE ON CURRENT DISK KEY RETURN(EXEC) TO RESUME? | The specified ISS component does not exist on the ISS disk(ette) in use. | Either touch SF'31 to obtain the System menu, or mount the correct ISS disk(ette) and touch the RETURN key. |
| RE-ENTER<br>nn xx nn<br>(ddd.) | During entry of a numeric value, the number entered falls outside of the acceptable upper and lower numeric limits (range) which are displayed. The (ddd.) indicates the number of digits and decimal position. | Re-enter a numeric value within the displayed upper and lower numeric limits. |

Table 1-3. Start-up and Menu Access Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| UNABLE TO OPEN STATION #nn. KEY RETURN(EXEC) TO RESUME? | The station file for this station number is scratched or insufficient room exists on the ISS disk(ette). This occurs after start-up software automatically attempted to create a station file. | Touch the RETURN key. Either enter a different station number or mount a different ISS diskette. |
| VP OPERATING SYSTEM RELEASE 1.9 (MINIMUM) REQUIRED, KEY RETURN (EXEC) TO RESUME? | ISS-5 requires the 2200VP Operating System (OS) be Release 1.9 or more recent. | Master Initialize the system and load Release 1.9 (or more recent) of the 2200VP OS. If not available, obtain Release 1.9 (or a more recent release) from Wang Laboratories, Inc. |

## 1.5 APPLICATION PROGRAM REQUIREMENTS AND ISS COMMON VARIABLES

Table 1-4 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 1-4. ISS Common Variables

| DESCRIPTION | NAME |
|---|---|
| Julian Date | Q1 |
| Gregorian Date | Q1$ (8 bytes) |
| Memory Size (K) | S |
| CRT Width (64,80) | S0 |
| Station Number (1-48) | S2 |
| CPU Type (3=VP, 4=MVP) | S3 |
| ISS Loading Address | S$ (3 bytes) |
| Printer Address | S$(1) (3 bytes) |
| Disk Addresses | S$(2) through S$(19) (3 bytes each) |
| Working Variable for Multiple Program Load | S1$ (64 bytes) |
| Flag for Menu to Load | S8$ (11 bytes) |
| Menu to Load | S9$ (11 bytes) |

## Application Program Requirements

Application programs running under ISS should adhere to the following requirements and recommendations:

1. All variables (numeric or alphanumeric, scalar or array) 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, the application program should, upon its completion, furnish a return to ISS by (1) executing a SELECT DISK xyy statement, where xyy is the ISS loading address, (2) providing the operator with an opportunity to remount the ISS disk(ette); e.g., a prompt, and (3) executing the statements LOAD and RUN to load 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 SF'31 (use a DEFFN'31 and a COM CLEAR statement).

5. With application programs using global program text, programming conventions applicable to the 2200MVP Operating System should be adopted. For instance, the currently selected global partition is determined via a SELECT @PART statement in the user's application program; such a statement allows DEFFN' statement subroutines in the selected global partition to be accessed via GOSUB' statements in the user's application program(s). To determine the partition size required by an application program, after the program has been loaded and run, touch the HALT key and the RETURN key and enter :PRINT SPACEK - SPACE/1024.

* As an alternative, the user program can (1) clear the application program variables, (2) set common variable S8$="SYSTEM", and (3) execute a LOAD DCT "ISS.002M" (the System menu).

# CHAPTER 2
# THE ISS UTILITIES

## 2.1 INTRODUCTION

The ISS utilities provide several standard functions necessary for a disk-based data processing environment. Categories of the functions available with the ISS utilities include Copy Functions, Programming Functions, Catalog Index Functions, and Special Purpose Functions.

Table 2-1 shows the categories of ISS utility functions relative to the elements of disk storage. Programming Functions apply only to program files, and the Copy Functions apply to both data files and program files (all files).

A brief description of each ISS utility appears in Chapter 1, Section 1.1.

Table 2-1. ISS Utilities and Categories

| FUNCTIONAL CATEGORY | STORAGE ELEMENT | UTILITIES |
|---|---|---|
| Copy Functions | all files | Copy/Verify |
| Programming Functions | program files<br>program files<br>program files<br>program files | List/Cross-Reference<br>Compression<br>Decompression<br>Program Compare |
| Catalog Index Functions | disk<br>disk<br>disk | Sort Disk Catalog<br>Reconstruct Disk Index<br>Alter Disk Index |
| Special Purpose Functions | all files<br>all files<br>data files | Create Reference File<br>Disk Dump<br>File Status Report |

## The ISS Utilities Menu

The Utilities menu, as shown in Figure 2-1, may be obtained following start-up either by specifying UTILITIES as the MENU TO LOAD or by touching the appropriate SF key in reply to the System menu.

```
SELECT UTILITY

                              ISS UTILITIES (STATION # = 1)
     ----------------------------------------------------------------------
      FN KEY        PROGRAM NAME                 FN KEY        PROGRAM NAME
     ----------------------------------------------------------------------
        00      COPY/VERIFY                        05      SORT DISK CATALOG
        01      CREATE REFERENCE FILE              06      DISK DUMP
        02      LIST/CROSS-REFERENCE               07      FILE STATUS REPORT
        03      COMPRESSION                        08      PROGRAM COMPARE
        04      DECOMPRESSION                      09      RECONSTRUCT DISK INDEX
                                                   10      ALTER DISK INDEX

                                                   31      SYSTEM MENU
```

Figure 2-1. The ISS Utilities Menu

To load one of the ISS utilities, touch the appropriate SF key. To load the System menu, touch SF'31.

## ISS Utility Default Values and Operating Similarities

For each ISS utility except Alter Disk Index, default values are maintained for each station in its respective station file. When an ISS utility is loaded, its default values are loaded from the appropriate station file. The utility's default values (if any) are displayed and modified exactly like the ISS start-up default values; that is, each default value appears to the right of a number which allows modification when it is entered in reply to the ENTER DESIRED FUNCTION prompt. If there are no default values for the chosen utility, entry of each value is requested beginning with value 1.

After modifying and/or entering the required values and verifying that the displayed values are correct, the user may save the currently displayed values as default values (SAVE DEFAULTS) before entering 0 (zero) to proceed. In addition to the default values, other prompts may appear requesting nondefault information or such manual actions as mounting a diskette.

If the utility default values for one or more disk addresses conflict with the start-up disk addresses, entry is automatically requested when that utility is loaded. Whenever a disk address is requested, valid start-up disk addresses are displayed.

There are two phases of ISS utility operation: the parameter entry phase and the execution phase. The final step in the parameter entry phase (except for the PART and INDIRECT modes where additional file names must be entered) is the appearance of a MOUNT DISK, KEY RETURN (EXEC) TO RESUME? prompt (INPUT DISK or something similar may follow the word MOUNT instead) after all parameters have been entered. Once the RETURN key is touched, the execution phase begins, and the prescribed task is performed. During both phases, SF'31 is available to close all files and return the ISS Utilities menu to the screen. Only during the parameter entry phase is SF'15 available, which returns the ENTER DESIRED FUNCTION (0=END) prompt to the screen, allowing modification of any of the default parameters.

## Methods of Multiple File Processing: The MODE Parameter

Certain ISS utilities support multiple file processing via the MODE parameter whose value determines which files are to be processed by the chosen utility. The multiple file processing utilities are the following:

Copy/Verify - copies and/or verifies each specified input file to a specified output file which may be created by the utility. Data or program files are acceptable.

List/Cross-Reference - lists and/or cross-references each specified input file. Only program files are acceptable.

Compression - compresses the contents of each specified input file and copies the compressed contents to a specified output file which may be created by the utility. Only program files are acceptable.

Decompression - decompresses the contents of each specified input file and copies the decompressed contents to a specified output file which may be created by the utility. Only program files are acceptable.

File Status Report - closes or outputs (prints or displays) the status of each specified input file. Only data files are acceptable.

Program Compare - compares the program text contained within each pair of specified files. Only program files are acceptable.

The Mode parameter's significance depends, in part, upon the acceptable file type (data or program) for the utility used. A disk device address specifies where all input files reside and a second disk device address specifies where all output files are to reside (if required by the utility). Each file is identified by a file name unique to all other file names on the same disk(ette). All input files must be active files (not scratched files).

There are four general Mode parameters applicable to most ISS utilities: All, Range, Part, and Indirect.

1. ALL indicates that all acceptable files on the input disk are to be processed without user-specification of file names; if the utility copies the input file, each output file name is assigned the corresponding input file name. Files names are not entered.

2. RANGE indicates that all acceptable files on the input disk whose file names fall within a user-specified alphabetic or alphanumeric range (upper and lower limits) are to be processed without user-entry of file names; if the utility copies the input file, each output file name is assigned the corresponding input file name. File names are not entered, but a lower and upper limit of file names are entered.

3. PART indicates that up to 25 input file names entered by the user are to be checked to ensure they exist and then processed; if the utility copies the input file, the user enters the corresponding output file name which is also checked. File names are entered, and the appropriate disk(s) will be requested via a MOUNT DISK, KEY RETURN(EXEC) TO RESUME? which allows on-line checking of entered file names.

4. INDIRECT indicates that a reference file resides on the same disk as the input files and contains the input file names and corresponding output file names (if required) to be processed, which were previously specified during operation of the Create Reference File ISS utility. A reference file may be created, edited, or printed using the Create Reference File utility. Several ISS reference files listed in Table 1-2 are included along with the ISS program files and station files.

When copying files, the input and output disk device addresses may be the same if the Part or Indirect mode is specified. When copying to the same disk, the All or Range mode must not be specified because the output file names (which are assigned) cannot be identical to the input file names if the same disk is used for both input and output.

The Copy/Verify utility, because it alone can process both data and program files, supports two additional Mode parameters. In addition to All files, all Data files or all Program files may be copied and/or verified. The DATA or PROGRAM mode is identical to the All mode, except that only data files or only program files are processed from the input disk.

Although the Program Compare utility does not copy files, the second program file of a pair to be compared is like an output file in the preceding discussion. For instance, the All or Range mode cannot be specified if the disk addresses of the first and second program files are the same; the result would be to compare each program file to itself because the second file name is assigned the name of the first file.

## Copying Files: Specifying Extra Sector Values

The Copy/Verify and Decompression utilities employ an Extra Sectors parameter whose significance depends upon the mode specified for these utilities; if the mode is All, Range, Data, or Program, the extra sectors value is assigned uniformly to all files copied. If the mode is Part, the extra sectors parameter becomes an operator-modifiable value for each file copied, and the operator can individually specify the number of extra sectors for each file.

Extra sectors describe the unused disk space within a file, that is, the number of sectors available for additional program text or data records between the (DATASAVE DC END statement or the END Disk subroutine) "END (end-of-data) control sector" and the "catalog trailer (end-of-file) control sector." (Please note that these terms are used throughout this manual to identify these two control sectors.) By specifying the extra sectors value as a large value for a file which has no extra sectors (such as a source program file), the output file will have sufficient extra sectors to accommodate additional program text or data records.

## Copying Files: Output Disk(ette) Requirements

Any output disk or diskette to receive copied files must have a catalog index (SCRATCH DISK statement). Also, a tab must be in place over the write protect hole.

## Use of Multiplexed/Multistation Files

Data files may be created as multiplexed/multistation data files by using the appropriate Disk subroutines (see Chapter 3). Such data files may be accessed (opened) in one of four access modes, and a file password may be required to access the file. The four access modes are Inquiry, Read Only, Shared, and Exclusive as further described in Table 3-1. Create Reference File employs Exclusive access to the specified reference file. All other utilities employ the Shared access mode when accessing specified input files. With the Shared access mode, application programs may be simultaneously updating records while that file is being copied or verified utility. Therefore, the output file may not reflect the file's contents at any one time (or may not verify) unless the user makes sure that updates will not be occurring during Copy/Verify operations on that file.

Entry of a password is required during the Disk Dump utility if the file is a multiplexed/multistation data file and the password is anything other than blanks (16 characters are allowed for the password). The output file is assigned the same password as the corresponding input file with the Copy/Verify utility.

Files on a multiplexed/multistation disk(ette) drive should be scratched only when no other stations are accessing those files. All ISS utilities assume the files to be accessed will not be scratched after their file names have been entered until processing has been completed.

Note that a DATASAVE DC END statement destroys that file's password and access table. The COPY statement, any form of the MOVE statement, or the Copy/Verify utility should be used for copying purposes. The END Disk subroutine should be used instead of the DATASAVE DC END statement.

### Error Messages and the Start-up Printer Address

During the entry of parameters in reply to prompts, any error message encountered appears on the station's screen. Once utility processing (execution) has begun, however, error messages are output to the printer address specified for this station. Error messages may not appear long enough for detection if address 005 is used for printing, and they are omitted if address 000 is used.

If the specified printer address does not correspond to an actual printer (e.g., blank), an error message appears when attempting to load the List/Cross-Reference utility. With the Create Reference File, Disk Dump, Sort Disk Catalog, Program Compare, and File Status Report utilities, should the user specify printed (list) output and the printer address does not correspond to an actual printer, an error message appears.

The printer is hogged during the output from List/Cross-Reference, and it is hogged if printer output is specified for the Sort Disk Catalog, Program Compare, File Status Report, and Disk Dump utilities. It is briefly hogged during Copy/Verify, Compression, and Decompression.

Error messages for all ISS utilities are listed in Table E-1.

## 2.2 THE COPY/VERIFY UTILITY

The Copy/Verify utility copies specified files on a file-by-file basis from one disk(ette) to the same or a different disk(ette). The copied output files may be verified. The specified files may be verified automatically following the copy operation for each file (Copy/Verify option) or the specified files may only be copied (Copy option) or only be verified (Verify option). The content of each input file up to and including the END control sector is copied into the output file. Input file sectors beyond the END control sector up to the next-to-last sector are not copied, but the user may specify the number of extra sectors to be included in the output files. The last two sectors in a file, the next-to-last sector (copied for compatibility with KFAM-7 conventions) and the catalog trailer (end-of-file) control sector, are copied first (unless a one sector file is copied). The next-to-last sector may be overwritten by the END control sector if 0 (zero) extra sectors is specified. If an END control sector does not exist, all input file sectors and the specified number of extra sectors are copied to the output file; an END control sector is not written. The Verify option compares the input and output files on a sector-by-sector basis.

The Output Option parameter -- either Add, Replace, or Add/Replace -- determines where the copied contents of the input file are to be written. If the Add option is specified, the input file's contents are copied into an output file created (cataloged) by the utility using the specified or assigned output file name which must be unique to the output disk. If the Replace option is specified, the input file's contents are copied into a previously cataloged, scratched or active output file whose file name is identical to the output file name specified or assigned. If the Add/Replace option is specified, the input file's contents are copied either into an output file created by Copy/Verify using the specified or assigned output file name if that name is unique to the output disk, or if the output file name exists on the output disk, the file whose name is identical to the output file name is copied into and overwritten. The output file is assigned the status (data or program file) of the input file and the password (if any) of the input file. With the Replace option, the sectors used by the input file plus the number of extra sectors specified must not exceed the number of sectors allocated to the output file to be replaced, or an error message is printed or displayed and that file is not copied.

The extra sectors value may be -1, 0, or a positive integer less than 250. If -1 is specified, the output file will have as many used and extra sectors as the input file. If 0 (zero) is specified, there will be no extra sectors in the output file. Otherwise, the number entered becomes the number of extra sectors in the output file.

The available Mode options include All, Part, Range, Indirect, Program, and Data as explained in Section 2.1. With the All, Range, Program, and Data modes, output file names are assigned the same name as the input file, all files are assigned the same number of extra sectors, and the input disk address (disk copied from) must not be the same as the output disk address (disk copied to). With the Part and Indirect modes, the output file name and number of extra sectors are assignable to each file, and the input and output disk addresses may be the same. With the Part mode, input file names entered are acceptable only if the file name specified is a cataloged and active file on the input disk; invalid file name entries are erased from the screen and the cursor is repositioned. Similarly, duplicate output file names are unacceptable. With the Indirect mode, all file names and extra sectors values are read from the specified reference file, which must reside on the input disk along with all input files. The specified extra sectors value (default parameter) is ignored with the Indirect mode only.

For backup disk copying, using the Copy/Verify utility with the ALL mode specified copies all active files from the input disk but does not copy scratched files (similar to the MOVE statement which can also be used); the COPY statement copies both active and scratched files.

When copying KFAM-7 files, the User File and Key File(s) should be copied. The KFAM utility Reallocate KFAM File Space is required following a Copy/Verify operation unless an extra sectors value of -1 was specified.

Please note that any KFAM User File should not be copied with an extra sectors value of 0 (zero), since the next-to-last sector which contains recovery information would be overwritten; otherwise, this sector is copied.

Error messages and recovery procedures are provided in Table E-1. The Utilities menu appears upon completion of utility execution.

## 2.3 THE CREATE REFERENCE FILE UTILITY

The Create Reference File utility allows the user to create a new reference file, modify (edit) a previously created reference file, and print the contents of a reference file. A reference file is a cataloged data file which contains multiple entries in the following form, per entry: input file name, output file name, and number of extra sectors. (However, only the input file name is actually checked, which allows special reference file use (Indirect mode) by the KFAM utility, Reset Access Tables.) Once created, a reference file can be used by any utility supporting the Indirect mode to specify the input file names, output file names (if applicable), and the extra sectors (if applicable) to be processed.

The Create option creates a new reference file, catalogs it, and allows selection of the files on the input disk to be included as input files from a listing of the input disk's catalog index. The operator specifies the reference file name. After specifying the input files, the output file names and extra sector values may be entered using operator-modifiable defaults for each. The specified number of entries determines reference file size, and the specified extra sectors default determines the displayed operator-modifiable extra sectors value provided for each entry.

The Edit option allows the entries in a previously created reference file to be modified. The Print option allows the contents of the reference file to be printed to the start-up printer address; a printer is required.

One of the parameters is called Type. Prior to ISS-4 and ISS-5, a reference file did not contain an extra sectors value for each entry. The Type of New corresponds to ISS-4 or ISS-5 reference files, whereas Old corresponds to a reference file created using a previous release of ISS. In most cases, the type of New/New should be specified to indicate the existing reference file (if created previously) and the resulting reference file are to be compatible with ISS-5 utilities. To update a reference file created using ISS-2 or ISS-3 to ISS-5 compatibilty, choose the EDIT option and specify the Type as Old/New. Create Reference File can read and write both the old and new type of reference file. ISS-5 reference files are multiplexed/ multistation data files with a password of ISS 4.0 REF.

### Maximum Number of Entries and Memory Requirements

The reference file consists of multiple entries which are loaded into memory and equated to table elements. As the number of entries increases, the amount of memory needed to contain this table also increases. The entered number of entries is automatically rounded upwards in multiples of 14. Table 2-2 lists the maximum number of entries for different memory sizes corresponding to 2200MVP partition sizes.

Table 2-2. Create Reference File Memory Requirements

| 2200MVP PARTITION | MAXIMUM NUMBER OF ENTRIES |
|---|---|
| 8K | 56 |
| 9K | 112 |
| 10K | 182 |
| 11K | 236 |
| 12K | 294 |
| 14K | 420* |
| 16K | 532 |
| 18K | 658 |
| 20K | 784 |

* Approximate maximum for a 16K 2200VP.

## Use of Reference Files by Utilities

The input file name, output file name, and extra sectors value in each entry are used by the Copy/Verify and Decompression utilities if the mode is Indirect, irrespective of the extra sectors value specified during Copy/Verify or Decompression operation. With Copy/Verify operation, extra sectors values of -1, 0, and positive integers up to 250 are valid; with Decompression, positive integers up to 250 are valid. Only the input and output file names are used with the Compression and Program Compare utilities; with Program Compare, the input file name corresponds to Input One and the output file name corresponds to Input Two. Only the input file name is used by the List/Cross-Reference and File Status Report utilities. The KFAM utility Reset Access Tables uses the input file name column for the User File name, the output file name column for the Key File/User File disk addresses (in the form xyy/xyy), and the extra sectors column for the Key File number (as described in Chapter 6, Section 6.11).

## General Operating Notes

After choosing the create, edit, or print option, the default parameters appear allowing specification of the file name, disk address, etc. With the print option, the reference file is printed after the default parameters have been accepted. With the create and edit option, a "window" of ten active files from the disk's catalog index is displayed under the Index column after accepting the displayed defaults, facilitating selection of files from the index list to become input files. Whether each file is a data (D) or program (P) file is also indicated. To the right of the Index column are the columns Input, Output, and X-sec. The Input column is where selected input file names appear. The Output column is where an operator-modifiable default output file name appears for operator verification or modification. The X-sec column is where the default number of extra sectors to be assigned to the corresponding output file appears for operator modification.

Also present on the screen is a cursor and, on the same line as the cursor, a double arrow and a four-digit number which indicates the entry number. The cursor and double arrow remain on the same line of the CRT screen. SF keys enable either the window to the index or the table of input/output file parameters to be moved, depending upon the column under which the cursor currently appears. The ten-file-name window to the index may be thought of as a continuous, rotatable circle with a blank line separating the beginning and end of the index.

There are two distinct operating phases: (1) selecting the input file names to be copied, and (2) specifying the output file names and extra sector values (file parameters) via operator-modifiable default values. In order to select the input file names from the index list, the cursor must be under the Index column; in order to specify the file parameters, the cursor should be under the Input column. SF'0 places the cursor in the Index column; SF'1 places the cursor in the Input column. The cursor is initially located under the Index column.

## Selecting Input File Names

The user first selects the input files from the index by (1) positioning a file name in the index window to be used as an input file to the double arrow and then (2) touching the RETURN key which moves the file name to the Input column, increments the entry number, and moves the index and file parameters forward (up) one file name. To move the index forward (up) one or five file names, touch SF'12 (→) or SF'11 (---→) respectively; to move the index backward (down) one or five file names, touch SF'13 (←) or SF'14 (←---) respectively. To move the index to the first five file names on the input disk, touch SF'7 (BEGIN). By repeatedly moving the appropriate index file name to the arrow and touching the RETURN key, all input files can be selected. In order for an input file name to be selected from the Index column, the current entry pointed to by the double arrow under the Input column must be all blanks. The input file name may be modified by the operator and changed to file name which exists on the input disk.

## Specifying File Parameters

After selecting all input files, touch SF'1 to move the cursor to the Input column. With the cursor in the Input column, to move the current file parameters forward (up) one or five file entries, touch SF'12 (→) or SF'11 (---→) respectively; to move the file parameters backward (down) one or five file entries, touch SF'13 (←) or SF'14 (←---) respectively. To move the current file parameters to the first five file names on the input disk, touch SF'7 (BEGIN). To move the current file parameters to the last five file names, touch SF'4 (END).

To specify the output file name for the input file name currently pointed to by the double arrow, touch the RETURN key. The input file name appears as a default in the Output column with the cursor positioned under the first character for operator modification. Modify the output file name value if required and touch the RETURN key. The default extra sectors value appears under the X-sec column. Modify the extra sectors value if required; when acceptable, touch the RETURN key. The cursor is now positioned under the next input file name and the file parameters shift forward (up) one entry. In this manner, specify the file parameters for all selected input files.

Should the list of file parameters be erroneous, the following SF keys are available:

1. SF'8 (ERASE) erases the current line of parameters including the input file name.

2. SF'9 (DELETE) deletes the current line of parameters like SF'8, but moves all following file parameters forward (upward) one entry to accommodate the deleted line. Entry numbers are automatically updated.

3. SF'10 (INSERT) inserts file parameters into the current entry position and moves all following file parameters backward (down) one entry to accommodate the insertion. Entry numbers are automatically updated.

4. SF'15 (RECALL) recalls accidentally erased characters previously in the current field if the current field is blank.

If additional input file names are to be selected, the user must position the file parameters such that the current file parameters are blank. Thereafter, the input file name may be selected by touching SF'0 to move the cursor to the Index column, moving the index forwards or backwards to the correct file name, and then touching the RETURN key. SF'1 can be touched to specify that file's parameters.

### Indicating Completion

After verifying all file parameters as being correct, touch SF'16 to indicate completion. A final check of input file names and output file names (not duplicate) is made; if an error is detected, the cursor is positioned under the erroneous file parameter facilitating operator correction. Touch SF'16 after making each correction. The prompt DO YOU WISH TO PRINT THE REFERENCE FILE? appears if all file parameters are acceptable. Enter Y to print the contents of the reference file; however, the start-up printer address must indicate a printer. Otherwise, enter N.

Error messages are described in Table E-1. The Utilities menu appears upon completion of utility execution.

## 2.4 THE LIST/CROSS-REFERENCE UTILITY

The List/Cross-Reference utility lists and/or cross-references specified program files on a file-by-file basis from the specified disk(ette). The List component may be automatically followed by the Cross-Reference component (List/Cross-Reference option) or the two components may be executed independently (List option or Cross-Reference option). Input program files must be free of all syntax errors in order to obtain meaningful results. Any protected program file is not acceptable as an input file.

The List component prints the contents of each program file and breaks up all multistatement lines into "decompressed" form identical to the LIST D form of the LIST command, whereby each BASIC-2 statement is printed on a separate line, with the exception of: (1) a REM statement which is not the first statement in a multistatement line always follows the previous statement on the same line, and (2) a % (Image) statement causes all characters including colons on that line to remain on that line.

The Cross-Reference component builds and prints three cross-reference tables: a list of each referenced line number and each line number which references it, a list of line numbers which contain each variable, and a list of each DEFFN' statement's line number along with line numbers referencing (via a GOSUB' statement) that DEFFN' statement. Each line number referenced and DEFFN' statement is printed in ascending numeric order; each variable referenced is printed in ascending alphabetic order. A summary is also output.

During the Cross-Reference component, an internal table is built as the input program is examined for DEFFN' and GOSUB' statements, variables, and line references. Should this internal table be filled before the entire program has been examined, three partially complete tables are printed, the table is cleared, and the same program is examined from the point the utility's table became exhausted. The end result is two sets of partial cross-reference tables, with each set complete for the portion of the program examined.

An input program may have the REM % form of the REM statement inserted to print titles in expanded print (available only with matrix printers). The following conventions must be observed:

1. To supply two blank lines between the REM % statement and the expanded print title, the following format is necessary with only one blank between the % and the first character of the title: REM% TITLE.

2. To supply a forms feed following the REM % statement and the expanded print title, thereby printing the expanded print title at the top of the next printer page, the following format is necesary with no blanks between the %, the ↑ (upward arrow), and the first character of the title: REM%↑TITLE.

Default parameters include: the number of spaces the printout is to be indented from the margin, from 1 through 10; the line length including the margin, from 70 through 128 (this depends upon the paper size); number of lines per printed page, from 10 through 55; and the mode of All, Part, Range, or Indirect.

NOTE:

There are certain statement forms which cause nonvariables to be referenced as variables, array variables to be referenced as scalar variables, and variables to be not referenced. Refer to Appendix C for a list of cross-reference exceptions.

## Operating Notes

Touching the P key stops printing at the end of the current print page. Touching the H key stops printing at the end of the current program file. In both cases, KEY RETURN(EXEC) TO RESUME? appears allowing one of the following: (1) the utility may be aborted by touching SF'31, (2) the utility may begin processing the current file from the beginning by touching SF'0, (3) the utility will print the cross-reference tables for the current file based upon the program text examined thus far by touching SF'1 (only if the Cross-Reference option was specified), or (4) the utility may continue from the point it stopped by touching the RETURN key.

Error messages and recovery procedures are described in Table E-1. The Utilities menu appears upon completion of utility execution.

## 2.5 THE COMPRESSION UTILITY

The Compression utility reduces the amount of memory required by a program and increases the speed of program execution. On a file-by-file basis, each specified input program file is read, put into compressed form, and copied to the specified output file. Any protected program file is not acceptable as an input file.

For maximum disk efficiency, no files should be created on the output disk by other stations while the Compression utility is being executed, which prevents the internal execution of the Free Unused Sectors subroutine incorporated into Compression and leaves the compressed output file with a greater number of extra sectors than requested.

### Compression Rules and Exceptions

Certain rules are observed internally by the Compression utility while compressing program text. These rules are as follows:

1. All REM statements are eliminated, with the exception of a REM statement contained within the first line of a program. REM statements may be located anywhere in a multistatement line.

2. All space characters (blanks) are eliminated, with the exception of blanks contained within quotation marks or an Image % statement.

3. Unnecessary line numbers are eliminated by appending as many BASIC-2 language statements as possible onto a single preceding line number. The maximum number of appended statements per line number depends upon the specified maximum number of bytes per statement line (Length option), either 180 or 256 bytes.

Certain exceptions to the above rules are observed by the Compression utility while compressing program text. These exceptions are generally necessary to preserve program integrity and are the following:

1. The first line in a program file is unaltered, including REM statements, which facilitates program identification and documentation. Statements following the first line are not appended onto the first line.

2. Certain BASIC-2 language statements are never appended onto a preceding line, namely DEFFN, DEFFN', FN, and Image % statements. Although these statements are not appended onto a preceding line, one or more following statements may be appended onto that line.

3. Certain statement lines are never appended onto a preceding line because of a particular BASIC-2 language statement contained within the preceding line.

   a. Any statement line referenced elsewhere within the program is never appended onto a preceding line. Statements which can reference a line number include GOTO, GOSUB, ON GOTO, ON GOSUB, IF THEN, ON ERROR, etc. If a REM statement is referenced, that REM is eliminated and a warning error message is printed or displayed.

   b. Any statement line preceded by a blank REM statement (REM followed by spaces) is not appended to a preceding line. However, because all REM statements are deleted during Compression, a subsequent Compression is likely to append that statement onto a preceding line unless a blank REM is inserted preceding that line prior to the subsequent Compression. Furthermore, should the user want a statement line neither appended to a preceding line nor have following lines appended onto it, blank REM statements should be inserted both preceding and immediately following the statement line to be unaltered during the next Compression.

   c. Any statement line following an Image %, GOTO, LOAD, RETURN, or ERROR statement immediately preceded by a colon (:) on a multistatement line (e.g., 100 DATALOAD DC A():ERROR GOSUB 1000) is never appended to a preceding line, regardless of the number of times the program is compressed.

## Operating Notes

Default parameters include the input and output disk addresses, the mode (All, Part, Range, or Indirect), and the Length option. If the Length option number is entered in reply to the ENTER DESIRED FUNCTION (0=END)? prompt, the current value (180 or 256) is replaced by the other possible value (flip/flop). If the Length option is 256 bytes, the line lengths are as long as possible, which reduces memory requirements and increases program execution speed more than 180 bytes. If the 256 option is specified, however, the size of the output program file may be larger than the size of the input program file because 256 bytes per line is not as disk-efficient as 180 bytes. For this reason, the output file is approximately 10% larger than the input file, and contains two extra sectors under most circumstances. If the program's text is likely to be recalled and edited on a Wang 2200T central processor, the 180 option must be specified.

Error messages and recovery procedures are described in Table E-1. The Utilities menu appears upon completion of utility execution.

## 2.6 THE DECOMPRESSION UTILITY

The Decompression utility reads each specified input program file, decompresses the program text, and copies the decompressed program text to a specified output file on a file-by-file basis. To decompress program text, the Decompression utility breaks up multistatement lines by assigning a line number one greater than the previous line number to the second statement and each following statement on a multistatement line. The decompressed statement lines are indented to facilitate rapid visual identification of executable (non-REM) statements versus REM statements and executable statements within a FOR...NEXT loop versus executable statements not within a FOR...NEXT loop. Any protected program is not acceptable as an input file.

The presence of certain BASIC-2 language statements within multistatement lines and the exhaustion of available line numbers result in all or part of an input multistatement line remaining a multistatement line in its decompressed form. The exceptions to the assignment of each statement to a unique, ascending line number are as follows:

1. Any REM statement which is not the first statement in a multistatement line is not assigned a line number; instead, it follows the preceding executable statement on the same line.

2. An ELSE clause (:ELSE) following any statement is not assigned a line number; instead, it follows that statement on the same line.

3. An ERROR statement which is not the first statement on a multistatement line is not assigned a line number; instead, the :ERROR statement and all following statements on the multistatement line follow the preceding executable statement on the same line.

4. As Decompression breaks up a multistatement line and assigns unique, ascending line numbers, there may be fewer available line numbers between the line numbers of the current multistatement line and next line than the number of statements to be assigned line numbers in the current multistatement line. If so, each statement is assigned a line number until line numbers are exhausted, and the last available line number becomes a multistatement line. Similarly, the greatest possible line number, 9999, may be a multistatement line.

To facilitate rapid visual identification of FOR...NEXT loops (defined by FOR...TO and NEXT statements) and executable statements versus REM statements, the following rules of indentation are used:

1. Each REM statement at the beginning of a line is indented one space from its line number.

2. Each executable statement is indented five spaces from its line number; however, an executable statement within a FOR...NEXT loop is indented two additional spaces per FOR...NEXT loop the statement is nested within.

Operating Notes

Default parameters are similar to the Compression utility, with the exception of the extra sectors value (from 1 to 250) which indicates the desired number of extra sectors to be contained within the output files. The actual number of extra sectors in each decompressed output file may not exactly equal the specified number of extra sectors, since the disk space required for the decompressed output program text depends upon the input program text. The input program text is examined only as it is decompressed, after the utility has opened the output file with a sector allocation estimated by the utility based upon input file size and the specified number of extra sectors. With the Indirect mode, however, this parameter is ignored; with Part mode, this parameter serves as a default value for the extra sectors value of each output file.

Error messages are described in Table E-1. The Utilities menu appears upon completion of utility execution.

## 2.7 THE SORT DISK CATALOG UTILITY

The Sort Disk Catalog utility prints or displays the contents of a specified disk's catalog index. Each file's name, beginning and ending sector addresses, and the number of used and extra (free) sectors is output in sorted order, which is determined by specifying one of the following options:

1. The Name option lists the file names in the catalog index according to alphabetic order (ascending ASCII collating sequence).

2. The Starting Sector option lists the file names in the catalog index according to the starting sector address of each file in the catalog area.

3. The Index Sector Sequence option lists the file names in the catalog index according to the order in which the file entries are stored in the catalog index.

Active files, scratched files, or both (All files) may be specified for the file name list. Disk catalog index information output includes the number of index sectors, the last sector address allocated to a file, the ending sector address of the catalog area, and the number of data and program files. The disk catalog report contains more information than the output of the LIST DC statement which it resembles.

The maximum number of file name entries which can be printed or displayed in a catalog index report depends upon the memory available, as listed in Table 2-3. If insufficient memory exists to list all catalog entries, an error message appears facilitating output of a partial list.

Table 2-3. Sort Disk Catalog Memory Requirements

| 2200MVP PARTITION | MAXIMUM NUMBER OF ENTRIES |
|---|---|
| 7K | 65 |
| 8K | 138 |
| 9K | 211 |
| 10K | 284 |
| 11K | 357 |
| 12K | 431 |
| 14K | 577* |
| 16K | 723 |
| 18K | 869 |
| 20K | 1015 |

*Approximate maximum for a 16K 2200VP.

## Operating Notes

The default parameter for output device can be either CRT or PRINTER. If the output device number is entered in reply to the ENTER DESIRED FUNCTION (0=END) prompt, the current value is replaced by the other possible value (flip/flop).

After the default parameters have been accepted, the prompt ENTER TITLE FOR LIST appears. With printed output, the title entered appears atop each printed page (in expanded print with matrix printers), along with the disk address, date, and page number. The start-up printer address must indicate a printer if printed output is chosen.

With CRT output, the catalog index report appears one screenful at a time. Touch the RETURN key to view the next screenful of the report.

After the catalog index report is printed or displayed, a prompt appears allowing (1) the report to be reprinted on the CRT, (2) the report to be reprinted to a printer, and (3) a return to the ISS Utilities menu.

## 2.8 THE DISK DUMP UTILITY

The Disk Dump utility prints or displays all or part of a specified disk or disk file. With printed output, either all records or records located within specified limits may be printed or displayed in one of three output formats. With user-interactive displayed output, the user views the contents of the current sector and may position it forward or backward, to the last sector before the END control sector or to any specified relative sector number via SF keys. The displayed current sector may also be printed.

With printed output, only sectors preceding the END control sector may be printed (except for program files where the END control sector is also printed). With displayed output, however, any sector on the disk may be displayed.

The sectors to be printed or displayed may be within a specified file (input mode of File) or within a specified range of absolute sector addresses (input mode of Range). With the File option, either all sectors in the file may be dumped or specified relative sector addresses. With the Range option, the lower and upper absolute sector addresses must be specified. All sector addresses are relative to 00000 as the first sector of the file (relative sector address) or disk (absolute sector address). Please note that with the combination of Data File Structure dump and the File input mode, instead of relative sector addresses, the logical record numbers are specified (relative to 00000); however, unless multiple sector records are present, logical record numbers and relative sector addresses are identical.

The three kinds of Disk Dump output formats are the Horizontal, Vertical, and Data Structure options. With CRT output, only the Vertical format is allowed. The Horizontal and Vertical formats print the two-digit ASCII hexadecimal value of each byte and the characters represented by the hexadecimal characters in the specified sectors. With the Horizontal dump, 32 bytes of hexadecimal values are followed by the corresponding 32 characters on each line; wide (14- by 11-inch) printer paper is required, and six sectors are printed per page. With the Vertical dump, each character and its corresponding two-digit hexadecimal value appear vertically in a column with three sectors printed per page; normal width (8 1/2- by 11-inch) printer paper may be used with printed output, and CRT output is also available. The Data File Structure dump is valid only for data files whose records were written using DATASAVE DC or DATASAVE DA statements. This dump requires wide printer paper. For each sector (physical record), each field's content, type (numeric or alpha), and length is printed. For multiple sector records, the physical record in each logical record is numbered.

The file name and page number appear atop each page. Each sector is identified by both its absolute and relative sector addresses. A period (.) is printed for hexadecimal values below 20, and an "at" sign (@) is printed for hexadecimal values above 7E.

## Operating Notes

Among the default parameters are input mode and output device, which alternate between FILE or RANGE and CRT or PRINTER respectively, when the corresponding number is entered in reply to the ENTER DESIRED FUNCTION (0=END) prompt (flip/flop). The File Name parameter is ignored if the input mode is Range.

After accepting the default parameters, a MOUNT DISK message appears. With printed output, to temporarily stop printing, touch the H key. After the remainder of the sector has been printed, KEY RETURN(EXEC) TO RESUME? is displayed and printing ceases. To continue printing, touch the RETURN key; to abort, touch SF'31.

With user-interactive displayed output, the first sector of the specified file or range is displayed after the MOUNT DISK message appears. The specified file's name (if the input mode is FILE) and sector address limits of the file or range are displayed on the top line of the CRT. Also, the absolute and relative sector addresses numbers are dynamically displayed as the user moves the displayed current sector.

The relative sector address is relative to the first sector of the file or range, depending upon the input mode. With CRT output, any sector on the disk can be displayed. Table 2-4 lists and describes the SF keys available for CRT output.

Table 2-4. Disk Dump SF Keys for CRT Output

| SF KEY | PURPOSE |
|---|---|
| SF'2 | Beginning at the current sector address, sectors are continuously displayed one-by-one in ascending sector address sequence (forward) until any key is touched. After touching any key, the current sector remains displayed until a valid SF key is touched. |
| SF'3 | Beginning at the current sector address, sectors are continuously displayed one-by-one in descending sector address sequence (backward) until any key is touched. After touching any key, the current sector remains displayed until a valid SF key is touched. |
| SF'4<br>(END) | Displays either the last sector in the specified range or the last sector (end-of-file catalog trailer sector) in the specified file, depending upon the input mode previously chosen. |
| SF'5 | Displays the sector preceding the END (end-of-data) control sector. The sector displayed is the last sector of data or program text, depending upon the file type. SF'5 is valid only if the input mode is Dump by File. |
| SF'7<br>(BEGIN) | Displays the first sector of the specified file or range. With a program file, the header sector appears. |

Table 2-4. Disk Dump SF Keys for CRT Output (continued)

| SF KEY | PURPOSE |
|---|---|
| SF'8 | Allows the user to enter the sector number to be displayed relative to the current sector address. Any negative or positive relative sector number may be entered as long as that sector does indeed exist on the disk in use. |
| SF'9 | Redisplays the most recently displayed sector. |
| SF'10 | Prints the currently displayed sector to the device identified by the ISS start-up printer address. |
| SF'11 (---►) | Moves the current (displayed) sector five sectors foward, increasing the current sector address by 5. |
| SF'12 (-►) | Moves the current sector one sector forward, increasing the current sector address by 1. |
| SF'13 (◄-) | Moves the current sector one sector backward, decreasing the current sector address by 1. |
| SF'14 (◄---) | Moves the current sector five sectors backward, decreasing the current sector address by 5. |
| SF'31 | Aborts operation of the Disk Dump utility and brings the ISS Utilities menu to the screen. |

Error messages are described in Table E-1. With printed output, the Utilities menu appears upon completion. With displayed output, touch SF'31 to obtain the Utilities menu.

## 2.9 THE FILE STATUS REPORT UTILITY

The File Status Report utility prints or displays the multistation access status of specified data files and can also close any specified data files accidentally left open on the specified disk. Designed especially for multiplexed/multistation data files, the utility prints certain access status information including each specified data file name and one of the following: (1) NOT OPENED if that file is not open to any station, (2) NOT A MULTIPLEXED FILE if that file is not a multiplexed/multistation file, or (3) the station numbers and respective access modes if the file is open to one or more stations. Optionally, the specified data files may be closed or a file status report printed for only one particular station number. This utility reads and may rewrite the catalog trailer (end-of-file) control sector used by the multiplexed/multistation Open/End/Close Disk subroutines to maintain a data file's access table. This access table information may erroneously indicate that a file is open because of operator accidents, power failures, and errors encountered during initial testing of application programs.

The mode is the first parameter used by the utility to determine the file names to be processed (specify either All, Part, Range, or Indirect). If a particular station number is specified (station number is greater than zero), either every specified multiplexed/multistation data file opened to the specified station number is closed (Output option 1) or the access status of each specified file open to the specified station(s) is listed, including whether or not other station numbers have the file open (Output option 2, 3, or 4). If the option for all station numbers is specified (station number equals zero), there are four available Output options: (1) close the specified data files for all possible station numbers (Close File), (2) print or display the file name and status of each specified data file relative to all station numbers (List Status any File), (3) print or display the file name and status of each specified data file that is a multiplexed/multistation file relative to all station numbers (List Status Mux'd File), and (4) print or display the file name and status of each specified data file that is a multiplexed/multistation file and is currently open to a station relative to all station numbers (List Status Open File).

CAUTION:

The option to close a file should be used with extreme caution by persons knowledgeable of multiplexed/multistation file operation and only when processing by the particular station number by all stations (whichever is specified) has been terminated for the file(s) to be closed. Also, never use the File Status Report utility to close a KFAM file except for KFAMWORK, a work file; instead, use the KFAM utility Reset Access Tables to close KFAM files accidentally left open.

### Operating Notes

The output device parameter can be either CRT or PRINTER. When the corresponding number for output device is entered in reply to the ENTER DESIRED FUNCTION (0=END) prompt, the current value is replaced by the other possible value (flip/flop).

With CRT output and the list status output option, each screenful appears accompanied by the prompt READY FOR NEXT DISPLAY, KEY RETURN(EXEC) TO RESUME? which allows the operator to touch RETURN after viewing the displayed information. After the entire report has been printed or displayed, or after all files have been closed, a prompt appears allowing the same function to be re-executed (including printing or displaying the report), or the ISS Utilities menu may be obtained.

Error messages are described in Table E-1.

## 2.10 THE PROGRAM COMPARE UTILITY

The Program Compare utility compares each specified pair of input program files and prints or displays differences in the program text contained within the two program files. Statement line numbers of the two program files are used as the basis of comparison. Messages are printed under the following conditions: if a statement line number exists in only one program file, if a statement line number exists in both program files but its program text content differs, if one program file ends before the other, and if all program files compared are identical. Program Compare ignores all REM statements and blanks except for those enclosed within quotation marks or part of an Image % statement. For example, the Program Compare utility considers the following statements identical:

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

The two program files in each pair are identified as Input One and Input Two. Messages identify differences between the pair of files being compared and are accompanied by appropriate labeling of disk addresses and file names. These informative messages are listed in Table E-1.

### Operating Notes

The output device parameter can be either CRT or PRINTER. When the corresponding number for output device is entered in reply to the ENTER DESIRED FUNCTION (0=END) prompt, the current value is replaced by the other possible value (flip/flop). The Error Limit parameter allows the comparison of the current pair of program files to be aborted if the specified number of errors is reached, from 1 through 999; an Error Limit value of 0 (zero) indicates no error limit. During program execution, the current pair of program files being compared may be aborted by touching SF'0.

With CRT output, each screenful appears accompanied by the prompt READY FOR NEXT DISPLAY, KEY RETURN(EXEC) TO RESUME?, allowing the operator to touch RETURN after viewing the information displayed. After all messages have been printed or displayed, a prompt appears allowing the same function to be re-executed with CRT or printed output, or the ISS Utilities menu may be obtained.

## 2.11 THE RECONSTRUCT DISK INDEX UTILITY

The Reconstruct Index utility is a recovery aid for a disk whose catalog index has been destroyed; e.g., by being accidentally scratched. The utility searches the specified disk for file control sectors written during catalog operations. Based upon the control sectors found, it attempts to reconstruct a catalog index for the files on the disk. Utility execution can take up to several hours, especially if numerous scratched files were located on the disk. The number of index sectors and highest sector address are required.

The utility assigns file names for all data files and for duplicate program file names in the format /*nnnn*/, where nnnn is a four-digit number beginning with 0001.

CAUTION:

Before running this utility, a backup of the disk must be made because the utility writes on the specified disk. This utility is a last resort in recovery procedures, and its success is entirely dependent upon the nature of the disk. It is not guaranteed to reconstruct the disk index.

### Operating Notes

There are no error messages. The Utilities menu appears upon completion of utility execution.

## 2.12 THE ALTER DISK INDEX UTILITY

The Alter Disk Index utility displays the contents of a specified disk's catalog index and performs certain special-purpose functions pertaining to the catalog index. A file may be renamed (assigned a different file name) which automatically makes the file status active, a scratched file may be activiated (its status changed from scratched to active), an active file may be scratched (its status changed from active to scratched), a specified file name may be searched for and its file usage parameters displayed, and the last file allocated on a disk may be removed which frees (deallocates) its disk space for allocation to other files.

CAUTION:

A backup copy of the disk to be accessed using the Alter Disk Index utility must be made prior to utility execution, since certain disk hardware errors may render the disk inaccessible. This utility should be used with great care and <u>only</u> by persons knowledgeable of 2200 disk operation.

## Operating Notes

The only parameter required is the disk address of the disk whose index is to be viewed or altered. A MOUNT DISK message appears after the disk address has been entered, and is soon replaced by the display format shown in Figure 2-2.

```
                                        INDEX SECTORS - aa
                   DISK ADDRESS - xyy   END CAT. AREA  - bbbb
                                        CURRENT END    - cccc

   INDEX SECTOR - 0

   FILE NAME              START          END             STATUS

   1
   2 filename   ====|     nnnn           nnnn             11
   3 filename             nnnn           nnnn             11
   4
```

Figure 2-2. Alter Disk Index Display Format

Figure 2-2 shows the information displayed by the Alter Disk Index utility, where: aa represents the number of index sectors (established via a SCRATCH DISK statement), xyy represents the specified disk address, bbbb represents the absolute sector address where the catalog area ends (established via a SCRATCH DISK statement), and cccc represents the absolute sector address of the last sector currently allocated to a file on this disk.

The number appearing to the right of INDEX SECTOR- dynamically reflects the current index sector position, whose contents are displayed under the column headings File Name, Start, End, and Status in a "window" of up to eight file name entries. The absolute sector addresses (nnnn) under the Start and End columns define the boundaries for the file whose name appears under the File Name column. (Instead of the nnnn under the Start and End columns, the words DEAD and SLOT may appear to indicate where an old file name of a since renamed file had once existed. A MOVE statement removes all "dead slots" and reorganizes the disk.) Each file has a status of two letters (11) where the first letter is either an A if the file is active or an S if the file is scratched; the second letter is either a D if the file is a data file or a P if the file is a program file. As shown in Figure 2-2, each file name position may not yet contain a file name entry. The cursor and double arrow (◄====►) indicate the current file name entry.

The disk index can accommodate up to 15 file name entries in the first index sector and up to 16 file name entries in each subsequent index sector. SF keys are provided to move the list of file entries forward (upward) or backward (downward) while the cursor and double arrow remain stationary; other SF keys allow the contents of a different index sector to be displayed. Note that a hashing algorithm is used to determine in what index sector a file name is placed; file names are stored neither sequentially nor alphabetically. Functions which should be used with extreme care are those performed by SF'0, SF'1, and SF'2. SF keys applicable to this utility are described in Table 2-5.

Table 2-5. Alter Disk Index SF Keys

| SF KEY | FUNCTION |
|---|---|
| SF'0 | The status of the "current file entry" is changed from active (A) to scratched (S) or from scratched (S) to active (A). The "current file entry" is pointed to by the double arrow. This function is performed automatically by touching SF'0; no prompt appears. |
| SF'1 | The File Name of the current file entry may be changed to a specified file name. After touching SF'1, the prompt ENTER NEW NAME appears. Either the function may be aborted by entering 0 (zero) or the new file name may be entered. If the new file name is entered, the new file name is assigned to the file whose information is currently displayed and appears elsewhere in the index. The old file name may be reused in that disk index. |
| SF'2 | The last file in the catalog area may be removed from the disk, which frees (deallocates) that file's sectors for use by other files and removes its entry in the index. After touching SF'2, the index sector containing the last file is located and displayed, accompanied by the prompt DO YOU WANT TO REMOVE THIS FILE? (Y/N). If Y is entered the file is removed; if N is entered, the function is aborted and the previously displayed index sector is redisplayed. |

Table 2-5. Alter Disk Index SF Keys (continued)

| SF KEY | FUNCTION |
|---|---|
| SF'3 | A specified file may be located in the index and its index sector displayed. After touching SF'3, the prompt ENTER FILE NAME appears. Either the function may be aborted by entering 0 (zero) or the file name to be located may be entered. When found, that file's index sector is displayed and that file becomes the "current file entry." |
| SF'4 (END) | Moves the "current file entry" to the last file within the current index sector. |
| SF'7 (BEGIN) | Moves the "current file entry" to the first file within the current index sector. |
| SF'11 (--->) | Moves the list of file entries forward (upward) five file entries within the current index sector, which repositions the "current file entry." |
| SF'12 (->) | Moves the list of file entries forward (upward) one file entry within the current index sector, which repositions the "current file entry." |
| SF'13 (<-) | Moves the list of file entries backward (downward) one file entry within the current index sector, which repositions the "current file entry." |
| SF'14 (<---) | Moves the list of file entries backward (downward) one file entry within the current index sector, which repositions the "current file entry." |
| SF'20 | Displays the contents of the last index sector. |
| SF'23 | Displays the contents of the first index sector. |
| SF'27 | Displays the contents of index sector located five sectors forward from the current index sector position. The last index sector is redisplayed, when reached, if SF'27 is repeatedly touched. |

Table 2-5. Alter Disk Index SF Keys (continued)

| SF KEY | FUNCTION |
|---|---|
| SF'28 | Displays the contents of the index sector located one sector forward from the current index sector position. The last index sector is redisplayed, when reached, if SF'28 is repeatedly touched. |
| SF'29 | Displays the contents of the index sector located one sector backward from the current index sector position. The first index sector is redisplayed, when reached, if SF'29 is repeatedly touched. |
| SF'30 | Displays the contents of the index sector located five sectors backward from the current index sector position. The first index sector is redisplayed, when reached, if SF'30 is repeatedly touched. |
| SF'31 | Aborts operation of the Alter Disk Index utility and returns the ISS Utilities menu to the screen. |

## 2.13 ISS UTILITIES ERROR MESSAGES

The ISS utilities provide error messages under certain conditions. Table E-1 lists all utility error messages and some typical BASIC-2 error messages (ERR 1nn form). Recovery procedures are also provided. Most error messages are accompanied by the audio tone.

Certain utilities use the $OPEN and $CLOSE statements to respectively set or release disk hog mode and printer hog mode. Should a power failure occur or utility execution be aborted by any means other than SF'31, the currently hogged disk drive or printer may be released by executing a $CLOSE statement that specifies that the appropriate device address is entered in the Immediate mode from the terminal which was executing the utility.

# CHAPTER 3
# THE SCREEN/DISK SUBROUTINES

## 3.1 INTRODUCTION AND OPERATING NOTES

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- 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 Screen subroutines are Data Entry, Date Routines, Position Cursor, Operator Wait, and the Print Routine. The Disk subroutines are Select/Validate Disk Addresses, Search Index, Allocate Data File Space, Free Unused Sectors, Limits Next, Open/Close Output, Open/Close Input, and the Multiplexed/Multistation Open/End/Close subroutines. The Translation Table subroutines are ASCII to EBCDIC, and EBCDIC to ASCII.

The subroutines chosen may be specified either as "global" or "nonglobal" (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 created (saved) at 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 each user-written 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 nonglobal is specified, only one program file is created containing Dimension statements and subroutines both of which incorporate 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 calling it. Parameter values passed from the GOSUB' statement are assigned to certain variables within the subroutine. If parameters (arguments) are required, "symbolic 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 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, 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, if the same subroutine is to be called 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 before calling the subroutine. File numbers are selected by executing a SELECT statement (such as SELECT #3/310), or by using the Select/Validate Disk Addresses subroutine (see Section 3.8).

## Reserved Statement Line Numbers

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 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.

## Choosing the Desired Subroutines

Following ISS start-up operation (see Chapter 1), 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)

In reply to the menu currently displayed, the user chooses the subroutines to be saved by touching the corresponding SF keys. As each subroutine is chosen, an asterisk (*) appears to the left of not only the chosen subroutine, but also any subroutines automatically included with the one chosen.

In addition to the SF keys available to choose the desired subroutines, the following SF keys are available:

1. To obtain the next menu of subroutines, touch SF'16. After touching SF'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.

   After choosing the subroutines desired, SF'16 should be touched again to allow the user to visually verify the subroutines chosen from the three menus.

2. To erase all subroutines chosen (indicated by asterisks) from all menus, touch SF'18. Because SF'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 nonglobal use, touch SF'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 System menu, from which the Applications menu may be obtained, touch SF'31.

2. To save the chosen subroutines for subsequent global (2200MVP only) use, touch SF '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 DIM 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 DIM 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 should (later) be loaded, a DEFFN @PART statement added to it, and it should then be resaved for subsequent use as a global program file.

   SAVING ROUTINES and STOP ROUTINES SAVED appear. To obtain the ISS System menu, from which the Applications menu may be obtained, touch SF'31.

### Estimating Partition Memory Size Requirements

In order to load a program file containing all ISS Screen/Disk subroutines, a 8.75K partition is necessary; however, in order to load and run a program file containing all ISS Screen/Disk subroutines, a 10K partition (8.9K memory) is necessary. These partition requirements are the same for both global and non-global subroutines. The variables required in the global output require a 2.75K partition to load and a 4K partition (2.8K memory) 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 the amount of unused (free) memory and the amount of memory actually necessary. For instance, after the program has been loaded and run, the Immediate Mode statement :PRINT SPACE K - SPACE/1024 prints the partition size needed by the program in K bytes.

## 3.2 DATA ENTRY (DEFFN'200)

The Data Entry subroutine, a Screen 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, (3) whether an alphanumeric entry is on a specified list of valid responses (table look-up), and (4) 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), a minus sign, a decimal point, and exponential characters (including the uppercase letter E) are valid entries.

A prompt can be displayed on line 1, by either the subroutine or the user program, and an operator-modifiable or unmodifiable default value 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 SF 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 the program calls the subroutine and the operator touches RETURN, any checks specified in the subroutine argument list are automatically 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?

2. With a numeric entry, is the number of digits to the left and right of the decimal point within the maximum length limit specified for each?

3. With an alphanumeric entry, is the length within the maximum limit specified?

4. With an alphanumeric entry, does the response conform to either the specified alphanumeric limits (without table look-up) or the list of acceptable responses (with table look-up).

If the entry is acceptable, a valid response is returned in numeric-variable Q9 for a numeric entry, or alpha-variable Q6$ for an alphanumeric entry. If a default value is used, the default value must be equated to the appropriate variable prior to each Data Entry subroutine call. The maximum entry field length for alphanumeric input is 64 characters. With numeric input, up to 62 characters can be entered; however, the 2200 system supports 13-digit precision for numeric field entry and calculations.

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 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.

Any error message displayed on Line 3 prior to calling the Data Entry subroutine is erased upon RETURN to the user's program from the Data Entry subroutine. The user's program can perform customized entry checking and, if invalid, can print the error message on Line 3 and branch back to the GOSUB' 200 statement. The error message remains displayed until the operator again touches the RETURN key to indicate completion after correcting the response.

<u>GOSUB' 200 Argument Format</u>

Transfer to the Data Entry subroutine occurs via the statement:

```
GOSUB' 200 (L$, H$, L1, R1, P$, T)
```

<u>Numeric Input Arguments</u>

Numeric input (symbolic variable) arguments are described below:

Range Check - L$ contains the <u>lowest</u> 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 <u>highest</u> 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 <u>left</u> of the decimal place.<br><br>R1 equals the maximum number of characters to the <u>right</u> 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$ contains the alphanumeric prompt displayed on line 1 of the CRT. The user's program may either let the subroutine display the prompt by setting the argument symbolic variable P$ equal to the prompt, or display the prompt prior to calling Data Entry and set argument P$=" " (blank) when calling the subroutine, to preserve the already displayed prompt. |
| Type of Entry | - | T determines the type of entry and whether a default value is used, as follows:<br><br>If T=-1, numeric input with an operator-modifiable, displayed default value occurs. The default value contained in alpha-variable Q6$ is displayed and may be modified if desired by the operator. Edit mode is activated.<br><br>If T=1, numeric input occurs without default.<br><br>If T=0, numeric input with a default value occurs without display. The default value contained in numeric-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 numeric-variable Q9 (and also in alpha-variable Q6$). |

<u>Alphanumeric Input Arguments without Table Look-up Checking</u>

Alphanumeric input arguments without table look-up checking are described below:

| | | |
|---|---|---|
| Limit Check | - | L$ contains the <u>lowest</u> acceptable alphanumeric string value of the entry.<br><br>H$ contains the <u>highest</u> 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 64 characters. If L1 equals zero (0), up to 64 characters will be accepted. R1 must equal zero (0). |

| | | |
|---|---|---|
| Type Of Entry | - | T may either equal 2, to indicate alphanumeric input without a default value, or equal 3, to indicate alphanumeric input with an operator-modifiable, displayed default value. If T=3, the default characters contained in alpha-variable Q6$ are displayed and may be modified. Edit mode is activated if T=3. |
| Prompt | - | P$ contains the alphanumeric prompt. The user's program may either let the subroutine display the prompt by setting the argument symbolic variable P$ equal to the prompt, or display the prompt prior to calling Data Entry, and set argument P$=" " (blank) when calling the subroutine to preserve the already displayed prompt. |
| Return | - | A valid operator reply is contained in alpha-variable Q6$. |

<u>Alphanumeric Input with Table Look-up Checking</u>

Alphanumeric input arguments with table look-up checking are described below:

| | | |
|---|---|---|
| Table of Valid Responses | - | L$ and H$ contain the valid responses. Both arguments are 64 bytes in length and are the two elements of an array which is searched character-by-character (via the MATSEARCH statement) with the operator response. Within the 128 bytes available for table look-up entries, each entry must equal the length of argument L1 and be padded with trailing spaces if necessary. If L$ and H$ are blank, no checking occurs. |
| Length Check | - | L1 equals the maximum number of characters for the field, up to 64 characters. R1 must be 1. |
| Type Of Entry | - | T may either equal 2, to indicate alphanumeric input without a default value, or equal 3, to indicate alphanumeric input with an operator-modifiable, displayed default value. If T=3, the default characters contained in alpha-variable Q6$ are displayed and may be modified. Edit mode is activated if T=3. |
| Prompt | - | P$ contains the alphanumeric prompt. The user's program may either let the subroutine display the prompt, by setting the argument symbolic variable P$ equal to the prompt, or the program may display the prompt prior to calling Data Entry, and set argument P$=" "(blank) when calling the subroutine, to preserve the already displayed prompt. |
| Return | - | A valid operator reply is contained in alpha-variable Q6$. The entry number in the array comprised of variables L$ and H$ is returned in numeric variable Q9. |

## 3.3 DATE ROUTINES (DEFFN'220,221,222,223,224,225)

The Date subroutines consist of a group of independently accessible Screen subroutines which facilitate the entry and use of dates. Dates may assume two forms: Gregorian and Julian. 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 $1 \leq MM \leq 12$.

DD is the day of the month $1 \leq DD \leq 31$, depending upon the month (including leap year for February).

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

$1 \leq DDD \leq 365$ whenever YY specifies a non-leap year, or

$1 \leq DDD \leq 366$ 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 Date subroutines 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.

If the date entered is acceptable, 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 numeric-variable U9; the subroutine is exited.

<u>Convert Date - Gregorian to Julian</u>

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 subroutine returns alpha-variable U9$ with the Gregorian date and numeric-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.

<u>Enter Date - Julian Form</u>

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 not entered in its 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 numeric-variable U9; the subroutine is exited.

<u>Convert Date - Julian to Gregorian</u>

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 numeric-variable U9 with the entered Julian date. No check is performed on J. A Julian date not in its proper form produces a Gregorian date with MM or DD outside the valid range.

Convert Julian Date to Proper Form

This subroutine converts any five-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 numeric-variable 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 numeric-variable U3 is returned as 65.

## 3.4 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 both the characters to the right of the new cursor position on the same line and 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. If desired, the PRINT AT function may be used instead of Position Cursor. A PRINT or LINPUT statement, or a call to the Data Entry subroutine typically 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. If E=-9E99 or E=9E99, the entire screen is cleared.

## 3.5 OPERATOR WAIT (DEFFN'254)

This Screen 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
```

## 3.6 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 CRT Line 3 when called to signal the operator of an entry error.

Transfer to the RE-ENTER subroutine is via the statement:

```
GOSUB' 255
```

## 3.7 PRINT ROUTINE (DEFFN'242)

The Print Routine subroutine is a Screen subroutine which prints a specified character 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.

## 3.8 SELECT/VALIDATE DISK ADDRESSES (DEFFN'205)

The Select/Validate Disk Addresses subroutine is a Disk subroutine which validates a specified disk address against a list of valid 2200 disk addresses and optionally selects that disk address to a specified disk file number (i.e., device table "slot"). The Select function is identical to that performed by the BASIC-2 language SELECT statement, thus allowing substitution of a Select/Validate Disk Addresses subroutine call for the corresponding SELECT statement. The list of valid disk addresses is an alphanumeric literal string consisting of all possible ISS start-up disk addresses.

Should the programmer wish to customize the disk address list to suit a particular disk configuration, the program file ISS.205S must be modified before the desired subroutines are chosen from the subroutine menus. This does not allow individual stations to determine the valid disk addresses for each day as in ISS start-up, but the list is set for all stations permanently unless the subroutine text is modified or ISS.205S is remodified and the subroutines are rechosen. To alter the disk address list in ISS.205S: (1) CLEAR all program text, (2) LOAD program file ISS.205S, (3) LIST line 8875, (4) carefully EDIT that line, which contains the literal string of 3-byte disk addresses, (5) SCRATCH the old ISS.205S program file, and (6) resave the new ISS.205S in place of the original file; e.g., enter SAVE DC T (), "ISS.205S".

The subroutine is called by:

GOSUB'205 (F,A1$, F1)

where: F is the file number (device slot) to be assigned the disk device address if the select option is specified (0-15).

A1$ is the disk device address in the "xyy" form to be validated and optionally selected.

F1=0 (zero) requests only validation of the specified address.

F1=1 requests both validation of the specified address and selection of the file number to the specified address, if it is valid.

The subroutine provides three possible return codes in alpha-variable Q$:

Q$="X" indicates argument F1 is invalid or the value of F is less than 0 (zero) or greater than 15.

Q$="I" indicates an invalid disk address.

Q$= blank indicates successful subroutine execution.

## 3.9 SEARCH INDEX (DEFFN'229)

The Search Index subroutine, a Disk subroutine, searches a disk catalog index for a specified file name. It returns the status of the file as active, scratched, or nonexistent, and indicates 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 (device slot).

N$ is the file name.

The numeric-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.
```

## 3.10 ALLOCATE DATA FILE SPACE (DEFFN'228)

This Disk subroutine catalogs (creates) a data file on any selected disk and allocates to it either (1) all available sectors between the current end of cataloged files and the end of the cataloged area, or (2) a specified number of sectors, to which it automatically adds two sectors to offset the two required system overhead (control) sectors. It checks the catalog index to ensure the uniqueness of the file name and allows a minimum acceptable number of sectors, not including control sectors, to be specified when all available disk space is to be allocated. Since the subroutine automatically accounts for system overhead sectors, the specified number of sectors is the number of extra sectors.

Allocate Data File Space is 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 number of sectors, not including the two file control sectors (END and catalog trailer sectors), to be used by the subroutine as a basis for opening the file. If S is equal to or greater than zero, the minimum number of sectors which must be available for the file to be opened is S plus 2, and all available disk space is allocated to the file, if opened. If S is less than zero, the absolute value of S plus 2 determines both the minimum number of sectors which must be available to open the file and the actual file allocation. For example, if S is 10, 12 or more sectors are allocated; if S is -10, then only 12 sectors are allocated.

There are three conditions sufficient to prevent the file from being opened. In the sequence of their evaluation they, and their return codes contained in alpha-variable R2$, are the following:

1. If the file name is the same as a cataloged scratched file, the return code R2$ is set to 3.

2. If the file name is the same as a cataloged active file, the return code R2$ is set to 2.

3. 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 occur, the file is opened and the return code R2$ is set to 0.

## 3.11 <u>FREE UNUSED SECTORS (DEFFN'227)</u>

This Disk subroutine examines a selected file in a disk catalog area. It reallocates those sectors (extra sectors) between the END (end-of-data) sector and the catalog trailer (end-of-file) sector as free sectors available to be allocated to other files. That file's catalog trailer sector is repositioned accordingly. The deallocation of sectors 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 an END control sector written by either a DATASAVE DC END statement or the Disk subroutine equivalent. If this subroutine is executed on a file without an END sector, that file is destroyed.

This subroutine is a counterpart to Allocate Data File Space.

The subroutine is called by:

```
SELECT #F/xyy
GOSUB' 227 (F, N$, S1)
```

where: yy 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:

1. If the file does not exist, the return code R2$ is set to 3.
2. If the number of extra sectors found in the file is less than or equal to the requested number of extra sectors, the subroutine returns 1 in R2$.

If none of the above conditions occur, 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.

## 3.12 LIMITS NEXT (DEFFN'226)

The Limits Next subroutine, a Disk 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:

```
SELECT #F/xyy
GOSUB' 226 (F,N$)
```

where: F = The disk file number.

N$ = The file name where the sequence is to begin. If argument N$=HEX(0000000000000000) or the file name in N$ is not cataloged, the scan begins with the first file in sequence.

xyy = The disk device address.

After return, this subroutine provides the following return values in alpha-variable R9$ and numeric-variable R:

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, upon return, the program should test for R=0. If R does not equal 0, the program loop can continue calling DEFFN'226.

## 3.13 OPEN/CLOSE OUTPUT (DEFFN'240,241)

These Disk subroutines open for output, and subsequently close, disk data files which utilize special-purpose 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<br>EOF-indicates end of file<br>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 software header and trailer records, which are not to be confused with the hardware END (end-of-dat·) and catalog trailer (end-of-file) control sectors, 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 cannot be opened for output again until this life span has expired.

Open Output

Before calling this subroutine, the numeric-variable Q1 should be equated to the Julian date (YYDDD form). 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 from 0 through 3 only.

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 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 numeric-variable Q1. Control returns to the application program with the current sector position being the first available file sector after the software header.

If the file name is cataloged and 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 from 0 through 3 only.

T$ is the software trailer indicator. T$ = EOF for end of file, or T$ = EOR for end of volume.

The subroutine writes the software trailer followed by the hardware END trailer.

If the file is the last file in the catalog area, the Free Unused Sectors subroutine is used 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.

## 3.14 OPEN/CLOSE INPUT (DEFFN'250,251)

These Disk subroutines open for input, and subsequently close, disk data files which utilize special-purpose 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 from 0 through 3 only.

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

Before calling the Close Input subroutine, the current sector position in the file must be at the software trailer.

The subroutine is called by:

```
GOSUB' 251(F)
```

where: F is the file number from 0 through 3 only.

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 increase the volume counter by one, and initiate the Open Input subroutine with the same file name and the new volume number specified.

## 3.15 INTRODUCTION TO THE MULTIPLEXED/MULTISTATION DISK SUBROUTINES

The Multiplexed/Multistation Open/End/Close subroutines provide file access control for both 2200MVP multistation use and multiple 2200VP access via multiplexed disk drives. (In previous releases of ISS, these subroutines were called "Multiplexed subroutines.")

Multiplexed/Multistation disk files provide additional file access security features not available with regular 2200 disk files. The security features require that user-supplied application programs always employ the Multiplexed/Multistation subroutines on data files, instead of the BASIC-2 Language DATALOAD DC OPEN statement and the DATASAVE DC OPEN, END, and CLOSE statements. Special information (namely an access table, the data file name, and a password) is maintained in a previously unused portion of the catalog trailer (end-of-file) control sector by these subroutines. Implementing a unique file password allows 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 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 section. All ISS utilities and subroutines are compatible with Multiplexed/Multistation files. KFAM utilities and certain subroutines use slightly modified versions of these Open/End/Close subroutines; it is recommended, however, that the appropriate KFAM subroutines be used to access KFAM files and not the Open/End/Close subroutines.

## File Password Use

A file is first created by the Open subroutine by what is called an "open new" operation. Whether or not a file password will be required by all stations attempting access to that 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 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. A file password may be up to 16 characters in length.

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.

## Converting to Multiplexed/Multistation Files

The conversion from a regular disk data file to a Multiplexed/ Multistation data file occurs when that file is opened using the Open subroutine ("open old"). Thus, by accessing the file, it is automatically converted to a Multiplexed/Multistation file; however, for the file to remain a Mulitplexed/Multistation file, the Open/End/Close subroutines must always be used instead of the DATALOAD DC OPEN and the DATASAVE DC OPEN, END, and CLOSE statements.

CAUTION:

The statement DATASAVE DC END destroys the the access table necessary for Multiplexed/Multistation files and also the file password (if any). Use only the Multiplexed/Multistation Open/End/Close subroutines when accessing Multiplexed/Multistation files and not the equivalent BASIC-2 language statements.

## 3.16 OPEN (DEFFN'217)

The Open subroutine (also called the Multiplexed/Multistation Open), by virtue of its arguments, allows a Multiplexed/Multistation file to be opened, the access mode defined, and a password (if any) used. When creating (cataloging) a new file or reusing the space occupied by a scratched file with the same file name, this is called an "open new." When accessing an existing 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 new" replaces the BASIC-2 language statement DATASAVE DC OPEN and similarly allocates file sectors and saves the necessary catalog index information. The "open old" is the equivalent of the DATALOAD DC OPEN statement and is for existing data files.

### Access Modes

The four access modes are called Inquiry, Read Only, Shared, and Exclusive. In the Exclusive mode, only the station with exclusive access may open the file. However, in order for that user to have exclusive access, no other station can have that file open. The Exclusive mode is automatically set by the Open subroutine 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.

When the Shared mode is 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 not usually 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 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 Open/End/Close subroutine use.

## Opening Two or More New Files

If two or more files will be opened as new files (open new), after checking that enough disk space exists for all files (use the Sort Disk Catalog utility), the following procedures are recommended:

1. Set disk hog mode ON (use a $OPEN statement or GOSUB' 215).

2. Call the Open subroutine with file parameters set for the first new file. Hold disk hog mode.

3. If successful, call the Open subroutine with file parameters set for the next new file. Hold disk hog mode.

4. Repeat step 3 for each new file. However, release disk hog mode (or use $CLOSE) after opening the last new file.

5. For each file, perform the following steps:

   a. Perform required processing.

   b. Move the current sector address to the appropriate end-of-data file location where the END control sector is to be written.

   c. Call the End subroutine.

   d. Call the Close subroutine.

## GOSUB' 217 Argument Format

After the file number has been selected to its disk device address (e.g., SELECT #3/310), transfer to the Open subroutine occurs via:

```
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. The file name is always within quotation marks in the argument list, unless an alpha-variable is used instead.

F contains the file (device) number, from 0 through 14.

C contains the station number, from 1 through 48.

S determines the type of open performed. An S greater than zero, indicates an "open new"; also, S equals the number of sectors to be allocated for this file, which should include two sectors for the END and the catalog trailer control sectors. S=0 indicates an "open old." S=-1 indicates a "reopen old" during which the access mode is changed. S=-2 indicates an "open old" but only if the file is a Multiplexed/Multistation file will subroutine execution be performed.

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 of up to 16 characters, if a password is required (within quotation marks). Password content is determined solely by that file's open new.

A1$ contains the disk address in the "xyy" form.

H=1 indicates that disk hog mode will be held following this call. H=0 indicates that hog mode is to be released (cancelled) following this call. The hog mode also may be released using GOSUB' 215, as described in Section 3.19.

NOTE:

Literal disk address A1$ must coincide with the address currently selected for the file number.

## GOSUB' 217 Return Codes

Alpha-variable Q$ contains a blank if subroutine execution was successful, or a return code as summarized in Table 3-1.

Table 3-1. Open Return Codes

| VALUE Q$ | INDICATES | CAUSE AND RECOVERY |
|---|---|---|
| blank | Successful open. | Continue |
| "A" | Access mode conflict (see "Access Modes" above). | 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). |
| "M" | Open old for only a Multiplexed/ Multistation file (S=-2) could not be performed because the file specified is not a Multiplexed/ Multistation file. | Check if value of argument S is correct. Check if the correct file was accessed (values F$,F,C,A1$). Retry with S=0 to convert this file to a Multiplexed/Multistation file or retry with correct file specified. |

Table 3-1. Open Return Codes (contiuned)

| VALUE Q$ | INDICATES | CAUSE AND RECOVERY |
|---|---|---|
| "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. |

3.17 END (DEFFN'218)

The End subroutine performs the function of the BASIC-2 statement DATASAVE DC END. As with DATASAVE DC END, upon calling the End subroutine, the current sector address must be at the location required for the END (end-of-data) control sector. The current sector address must not be at the catalog trailer (end-of-file) control sector when this subroutine is called.

Transfer to the End subroutine is via the statement:

```
GOSUB' 218 (F$,F,A1$,H)
```

where: F$ is the file name (within quotation marks).

F is the file (device) number, from 0 through 14.

A1$ is the disk address in the xyy form.

H=1 indicates that disk hog mode is to be held or obtained following this call. H=0 indicates that hog mode is to be released (cancelled) following this call.

Alpha-variable Q$ contains a blank if subroutine execution was successful. Otherwise, Q$ equals either "S" or "F", both of which indicate the file is full and it should be copied using the Copy/Verify utility or MOVE statement to increase the number of extra sectors in the file. Q$="S" indicates that the END control sector was successfully written in the next-to-last sector of the file, but it warns that the number of extra sectors in the file is now 0 (zero). Q$="F" indicates that the END control sector was not written, since the END control sector would have overwritten that file's catalog trailer (end-of-file) control sector. The return code of Q$="F" corresponds to an ERR D81 BASIC-2 Language error.

## 3.18 CLOSE (DEFFN'219)

Closing a file promptly is important because of possible access mode conflicts. Once access to a file is no longer needed, that file should be immediately closed, especially if it was opened with Exclusive access.

If necessary, a file may be left open for later processing. An unlimited number of files can be open for one station because station file status is maintained in the catalog trailer (end-of-file) control sector 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 is not recommended and may be recovered from by using the File Status Report ISS utility.

Transfer to the Close subroutine occurs via the following statement:

```
GOSUB' 219 (F$,F,C,A1$,H)
```

where: F$ is the file name (within quotation marks).

F is the file (device) number, from 0 through 14.

C is the station number, from 1 through 48.

A1$ is the disk address in the xyy form.

H=1 indicates hog mode is to be held following this call. H=0 indicates hog mode is to be released (cancelled) following this call.

There are no return codes.

## 3.19 SET/RELEASE HOG MODE (DEFFN'215)

If an 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.

**NOTE:**

**The Set/Release Hog mode subroutine uses device (file) slot #15. If hog mode will never be used by Open, End, Close, and Set/Release Hog mode subroutine calls, device slot #15 is available for use by that application program.**

## 3.20 TRANSLATION TABLE SUBROUTINES (DEFFN'201, 202)

The Translation Table subroutines assign specific sets of hex codes to an alphanumeric array so that it may be used as a translation table with the BASIC-2 statement $TRAN. The subroutines do not actually accomplish the translation; they merely initialize the array Q9$(). 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 |

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.

Assuming that alpha-variable 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
```

The characters supported for ASCII to EBCDIC and EBCDIC to ASCII translations are provided in Appendix D.

# CHAPTER 4
# THE SORT-4 DISK SORT SUBSYSTEM

## 4.1 INTRODUCTION

SORT-4 is a subsystem for sorting records contained within a disk data file. A user-written setup program provides the parameters for the sort and loads SORT-4 software. It eliminates the lengthy operator/screen dialog otherwise required for the entry of the sort parameters, although the user's setup program may request operator-keyed input of appropriate parameters. When sorting is complete, SORT-4 can load a specified application program module and can be used as a subsystem to an application program.

SORT-4 requires little operator attention but must be run either in a 2200VP or a 2200MVP foreground partition (currently attached to a terminal). With a 2200VP central processor, at least 16K memory is required. With a 2200MVP central processor, SORT-4 does not access a global partition and requires at least between 9K to 12K to run, depending upon the input file type (see Table 1-1).

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 operates in a 2200MVP multistation or multiple CPU disk multiplexed environment under ISS conventions, using the Multiplexed/Multistation Disk subroutines.

3. Six input file formats are supported:

    a. An ordinary cataloged data file,

    b. A BAS-1 data file,

    c. A data file opened and closed with ISS Open and Close Output/Input 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 consist of up to 10 sort key fields. Sort key fields 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 4-2 and the text following Table 4-2.

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 4.12.

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 Copy/Verify (Indirect Input mode). The name of the reference file is SORTREF4.

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 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 successfully 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 setup phase, start, processes setup parameters. |
| SORT400B: | Overlay SORT4 if KFAM input. |
| SORT400C: | Required for multistation files. |
| SORT401A: | Setup, continued. Process record format and sort key specifications. |
| SORT402A: | Setup, 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. |
| SORTREF4: | ISS Copy/Verify Reference File. |

## Minimum System Configuration

Station requirements for a 2200MVP central processor include at least a 9K partition for sorting a sequential (non-KFAM) file and 11K to 12K partition for KFAM files, depending on the size of the setup program. With a 2200VP, 16K memory is required. A disk or diskette is also required. SORT-4 programs require about 270 sectors.

## 4.2 WRITING THE SETUP 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 setup 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" setup program appears in Table 4-1 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 setup program. However, if a statement line does not have a default value or if the default value is not acceptable for a particular sort operation, that statement line must be included in the user's setup program either in the form shown in Table 4-1 or by use of an INPUT or LINPUT statement if that parameter must be decided by the operator at run-time. 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 4-1, 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 setup module. Variables and their values are critical to the successful execution of SORT-4; line numbers need not be the same as those listed in Table 4-1, but must be within the required ranges. Sample SORT-4 setup programs are provided in Section 4.19.

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. By using the parameters for multistation use in the setup program, the files will be handled by SORT-4 using the ISS Multiplexed/Multistation Disk subroutines. This is particularly important if the output file is a cataloged, Multiplexed/Multistation file because the output file's access table and password will be destroyed if multistation use is not specified (by the DATASAVE DC END statement).

Table 4-1. SORT-4 Master Setup 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$(1)21<br>(This line is necessary to define any arrays referred to below. It may be cleared when SORT4 is loaded because SORT4 defines all necessary arrays.) | none | |
| 179 | LOAD DC (device) "SORT4" 10, 179 | none | |
| | Machine configuration, see Section 4.5 below: | | |
| 3400 | M = memory size, K bytes. For a 2200MVP specify M = SPACEK + 2. For a 2200VP, specify M = SPACEK.<br>default = 16, 2200VP; or<br>S if under ISS | - | 4.5 |
| 3405 | MO$(1) = "table of device addresses"<br>default = "310320330350B10B20B30" | - | 4.5 |
| 3410 | SELECT #0 (first device address), #1 (second device address), etc.<br>default = SELECT #0 310, #1 320, #2 330, #3 350, #4 B10, #5 B20, #6 B30 | - | 4.5 |
| | Sort specifications: | | |
| 3415 | F = input file format<br>0 - unlabeled sequential<br>1 - BAS-1 labeled sequential<br>2 - ISS labeled sequential<br>3 - KFAM-3<br>4 - KFAM-4<br>5 - KFAM-5 or KFAM-7 | none | 4.3 |
| 3420 | N$(1) = input file name | none | |
| 3425 | F$(1) = input device address<br>(Hog mode address if multistation) | none | 4.6 |
| 3430 | P$(1) = input file password, if any | blank | 4.7 |
| 3435 | J = Key File number (KFAM only) | 0 | |

Table 4-1. SORT-4 Master Setup Program (continued)

| LINE | CONTENTS | DEFAULT VALUE | SEE SECTION |
|---|---|---|---|
| 3440 | F$(2) = Key File device address (KFAM only) (Hog mode address if multistation) | blank | 4.6 |
| 3445 | B = records per block | 1 | 4.4 |
| 3450 | B$ = blank, records written in DA or DC mode, or = "B" records written in BA mode | blank | 4.4 |
| 3455 | F$ = record format<br>blank = not packed<br>"A" = packed array<br>"P" = fixed length packed<br>"V" = variable length packed<br>"T" = TC format variable length packed records | blank | 4.4 |
| 3460 | N6 = maximum length of variable portion of record (variable length only) | 0 | 4.4 |
| 3465 | A$(1) through A$(4) = description of packed fields in record | blank | 4.4 |
| 3470 | I$ = reserved for future use | | |
| 3475 | D = starting block # to be sorted (sequential files only) | 1 | 4.9 |
| 3480 | L$ = "number of blocks of records to be sorted" or "ALL" (always in quotes) (sequential files only) | ALL | 4.9 |
| 3485 | A$ = starting key to be sorted (KFAM only) | HEX zeros | 4.9 |
| 3490 | E$ = limiting key to be sorted (KFAM only). Records up to, but not including, this key will be sorted. | HEX F's | 4.9 |
| 3495 | K = number of sort key fields | none | 4.8 |

Table 4-1. SORT-4 Master Setup Program (continued)

| LINE | CONTENTS | DEFAULT VALUE | SEE SECTION |
|---|---|---|---|
| 3500 | Per Sort key field, n: | | |
| | K(n) = sequence number of key field | none | 4.8 |
| | B(n) = starting byte (if partial field) | 1 | 4.8 |
| | N(n) = number of bytes (if partial field) | all | 4.8 |
| | STR(X9$,n,1) = HEX(01) if descending sort on this field; HEX(00) if ascending | HEX(00) | 4.8 |
| 3520 | N$(3) = sort work file name | none | 4.12 |
| 3525 | F$(3) = sort work file device address (Hog mode address if multistation) | none | 4.6 |
| 3530 | P$(3) = sort work file password, if any | blank | 4.7 |
| 3540 | P8 = maximum number of records to sort | * | 4.12 |
| 3550 | N$(4) = output file name | none | 4.13 |
| 3555 | F$(4) = output file device address (Hog mode address if multistation) | none | 4.6 |
| 3560 | P$(4) = output file password, if any | blank | 4.7 |
| 3565 | C$ = "Y" output file cataloged<br>"N" output file not cataloged<br>"W" use work file for output (tag sort only) | Y | 4.13 |
| 3570 | P7 = number of sectors in output file | * | 4.13 |
| 3575 | P8$ = "K" to force key sort<br>"R" to force full-record sort<br>blank: lets program decide between K and R, or "T" tag sort | blank | 4.10 |
| 3580 | D$ = "D" deferred mounting of output disk (full-record sort only) | blank | 4.13 |
| 3585 | S2 = station number (required if any disk device is multistation) | 0 | 4.5 |

*Default value described in the specified section.

Table 4-1. SORT-4 Master Setup Program (continued)

| LINE | CONTENTS | DEFAULT VALUE | SEE SECTION |
|---|---|---|---|
| 3590 | F$(5) = device address, SORT-4 program modules (Hog mode address if multistation) | none | 4.6 |
| 3595 | G$ = "name of special input procedure" (blank if none) | blank | 4.14 |
| 3600 | M4 = number of bytes occupied by special input procedure | 0 | 4.14 |
| 3605 | M$ = "name of program to load following the sort" (if blank, STOP) | blank | 4.15 |
| 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 | 4.15 |
| 3615 | F$(6) = device address for user programs G$ and M$ (Hog mode address if multistation) | blank | 4.6 |

## 4.3 INPUT FILE FORMAT REQUIREMENTS

One of the parameters in the setup 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." |
| 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. SORT-4 accesses KFAM files in logical key sequence. |
| 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. SORT-4 accesses sequential files in sequential physical record order. |
| 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

This general file format requires a cataloged disk file with an END (end-of-data) control sector (DATASAVE DC END). 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, 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 software header label. The first field of this label is alpha and contains HDR; the second field is alpha and contains the file name, which should match the input file name defined in N$(1) from line 3420 in Table 4-1. 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 4.4). 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 an END (end-of-data) control sector (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 Open and Close Output/Input Disk subroutines (see Chapter 3), 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 an END (end-of-data) control sector (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 4.4 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 END control sector (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, since deleted records are not indexed in the Key File.

KFAM-5 and -7 files contain an END control sector, and can therefore be specified as sequential files (file format 0) if a special input procedure (see Section 4.14) is written to exclude deleted records (and inactive records, if blocked) from the sort, assuming all deleted records contain a hexadecimal (FF) in the first byte of the KFAM key. SORT-4 processes sequential files faster than KFAM files, since it need not access the Key File.

## 4.4 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 4-2.

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 |

In discussing SORT-4 record types, the following terms apply:

1. A read field is any field (variable) contained in a DATASAVE statement argument list when the record is written to disk. Read fields, as stored on disk, are separated by control bytes (except for DATASAVE BA statement records, or BA mode). SORT-4 need not unpack read fields in an input record, since the fields are the ones read from or written to disk.

2. A packed field is any field (variable) which is packed into a read field before the record is written to disk via a DATASAVE statement. Since a packed field, as it resides on disk, is not separated from other packed fields by control bytes, it may be necessary to define the record so SORT-4 can unpack the packed fields in an input record.

3. A sort key is composed of one or more sort key fields, which collectively determine output record order. Each sort key field may be specified as an ascending or descending sort key field for the character-by-character comparisons based upon the ASCII collating sequence performed on all input records by SORT-4. The highest-order (primary) sort key field determines output record order for all records except those whose highest order sort key field contents are identical (duplicate). In this latter case, the next high-order (secondary) sort key field is used to examine input record contents, etc.

To differentiate between read fields and packed fields, the following example is provided. In the two statements which follow, alpha array P$() is a read field, and variables M$, M1$, M2$, and M3$ are packed fields.

```
$PACK (F=A$) P$ () FROM M$, M1$, M2$, M3$
DATASAVE DC #3, P$()
```

## 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 A0$(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.

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:

1. Create Wang signed packed fields always with an odd number of #'s following the sign.

2. 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 DC or DA mode where control bytes define fields, 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 four records per block, each record containing an array element of A$(), B(), and C$().

For records written in BA mode (without control bytes), a record format other than "blank" must be specified.

If the packed numeric fields in an array format record are not sort key fields, the record description in array A$() may be omitted, and partial fields to be used as sort keys may be defined in N() and B(). Conversely, if any of the sort key fields are packed numeric fields, the array A$() must be provided in the setup program.

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.

<u>F$ = "V", Variable Length Packed Records</u>

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 one or two 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 <u>must contain a fixed portion</u> 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.

<u>F$ = "T", Telecommunications (TC) Variable Length Packed Format</u>

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.

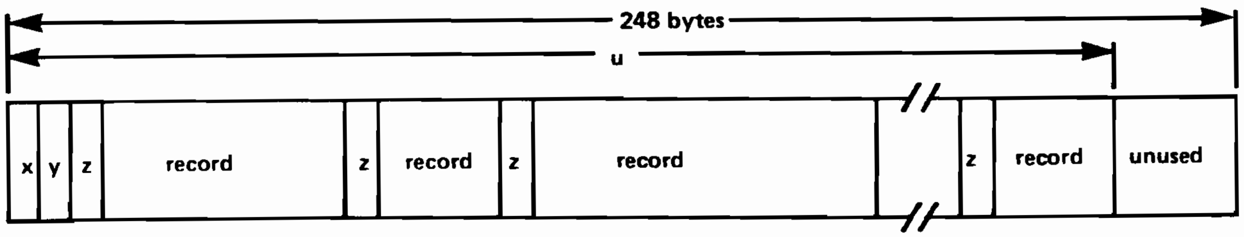
The TC format consists of variable length records contained within a 248-byte block including several control bytes. It is the user's responsibility to ensure that one or several entire variable length records are contained within this block. Individual fields written to disk as variable length records 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 an END (end-of-data) control sector (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 a 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 4-2 provides a cross-reference of record/file formats supported by SORT-4.

Table 4-2. 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. |

* F$ = "blank" is not valid for BA mode records. Use one of the other record formats.

Comments on Table 4-2

Input record formats and file formats are described in Section 4.3 and Section 4.4. The following notes apply to the combinations of KFAM file formats and SORT-4 input record formats described in Table 4-2.

1. 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). Certain KFAM record types apply to certain SORT-4 record formats (F$) as noted below and previously in Table 4-2.

   a. KFAM record type "A" always corresponds to F$="blank".

   b. KFAM-5 and KFAM-7 record type "B" always corresponds to F$="P"; B$ must equal "B" in the setup program.

   c. 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.

   d. KFAM record type "M", DC mode, corresponds to F$="blank" if records are unpacked, or F$="P" if records are packed.

   e. KFAM record type "M", BA mode, is not sortable because KFAM type "M" implies more than one sector per record (multiple sector record).

   f. 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 setup program.

2. 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:

   a. The record length is less than one sector, and records (if blocked) are blocked in a fixed block of exactly one sector.

   b. 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 sort key fields. 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 4-3 below.

Table 4-3. 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 four 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 setup 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 setup 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 setup 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 eight bytes, followed by a transaction code of two 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)
```

## 4.5 MACHINE CONFIGURATION

The following explanations of SORT-4 requirements and conventions are referred to in Table 4-1 by section number. File formats and record formats (including use of array A$() to define the fields in a record) were previously discussed in Sections 4.3 and 4.4.

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. Memory size is usually indicated by 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.

SORT-4 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 setup program. If running in a multistation or disk-multiplexed 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 not running in a multistation environment, or if all stations are the same size and access not more than a total of seven disk devices, it is probably better to change the machine configuration in the SORT4 module.

The table of device addresses, M0$(1)21, allows up to seven three-byte 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, the second as #1, and so forth. File numbers in the device table are linked to devices, rather than files. Hog mode addresses should <u>not</u> 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"
```

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 through 16 when sorting a KFAM file; otherwise, 1 through 48). |
| S | = | memory size (With a 2200MVP, use of M = SPACEK + 2 is recommended instead.) |
| S$ | = | system disk (ISS loading address), #0 |
| S$(2) thru S$(18) | = | other disk devices. |

Disk devices can be copied to the table as follows:

```
3405 MO$()= STR (S$,,3) & S$()
```

Note that SORT-4 allows only 7 disk devices.

To set the machine configuration for an individual sort, code the setup module as shown in lines 3400-3410, in Section 4.2, or use the following:

```
3405    MO$(1) = "table of xyy disk device addresses"
3410    FOR I  = 0 TO 6:  SELECT #I   STR(MO$(),I*3+1,3)  :NEXT I
```

## 4.6 DISK DEVICE ADDRESSES AND MULTISTATION OPERATION

To indicate multistation or disk multiplexed operation, 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$(), not in the table of device addresses, MO$(), described in Section 4.5.

If two or more files are on the same device, then the device addresses must be consistent, either multistation (hog mode) or not, as the case may be.

Files on a multistation disk (multistation files) are opened and closed using the ISS Multiplexed/Multistation Disk subroutines (described in Chapter 3), as follows:

The input file is opened in the Read Only access 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 open or 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 (SF'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 touching SF'31 to ensure that all multistation files are closed.

## 4.7 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.

Passwords are ignored on files not designated as multistation.

## 4.8 SORT KEY FIELDS

Up to ten 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 4.4. 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 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.

## 4.9 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 five 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 FF's, cause the entire KFAM file to be sorted.

## 4.10 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 (two-byte sector address and two-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. Many factors influence whether a full-record sort or a key sort is more efficient for a particular application, including record length, sort key length, and the amount of available memory. In general, either let SORT-4 decide or experiment with both types of sorts for a particular application.

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:

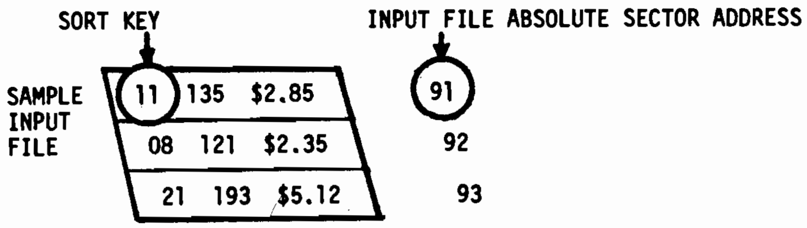
| | |
|---|---|
| Bytes 1,2: | Absolute sector address (hex) of corresponding input record. |
| Bytes 3,4: | For array type or packed array type (F$ = blank or "A"), pointer to record within block (hex) in byte 4. For packed records (F$ = "P", "T", or "V"), pointer to starting byte of record within block 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 4.15).

   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 4.13). 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 4-1 illustrates the basic content of three sample input records during each pass 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. 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 key sort output and full-record sort output are identical, and differ greatly from tag sort output.



| PASS | TAG SORT | KEY SORT | FULL-RECORD SORT |
|---|---|---|---|
| 1 | 4-BYTE POINTER<br>9100 11<br>9200 08<br>9300 21 | 9100 11<br>9200 08<br>9300 21 | 11 + RECORD IN BUCKETS<br>08 + RECORD IN BUCKETS<br>21 + RECORD IN BUCKETS |
| 2 | 9200 08<br>9100 11<br>9300 21 | 9200 08<br>9100 11<br>9300 21 | 08 + RECORD IN BUCKETS<br>11 + RECORD IN BUCKETS<br>21 + RECORD IN BUCKETS |
| 3 | 9200<br>9100<br>9300 | 08 121 $2.35<br>11 135 $2.85<br>21 193 $5.12 | 08 121 $2.35<br>11 135 $2.85<br>21 193 $5.12 |

Figure 4-1. SORT-4 Sample Operation on Input Records

## 4.11 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.

## 4.12 THE SORT WORK FILE

The sort work file must be cataloged as a disk file prior to running the SORT-4 setup 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 4-3 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 4-3 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, three 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 4.14).

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 approximately indicate 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 setup 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.

## 4.13 <u>THE OUTPUT FILE AND DEFERRED MOUNTING</u>

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 4.10 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 file 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 4.12).

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 setup 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.

SORT-4 writes an END (end-of-data) control sector following the last sector of live data in the output (or work) file. The user program(s) which processes the sorted output file should use an IF END THEN statement, and, if records are blocked, test for unused records in the last sector containing live data which are "padded" by SORT-4 to indicate that they are not valid records. Padded records fill any unused record positions in the last block. SORT-4 generates high values for both ascending keys and descending keys.

The padding procedure for the various record formats are as follows:

F$ = "0" 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.

## 4.14 SPECIAL INPUT RECORD SELECTION PROCEDURE

The user may specify a special input procedure to be overlaid in Pass 1 of the sort. The user's program text in the file indicated as G$ in the setup program is added to 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 file may be opened during the input processing phase; during this phase, input records are selected for sorting typically based on certain user-defined logical relationships between fields. If a special input procedure is used, the user's setup program must equate scalar variable M4 (Line 3600 in Table 4-1) to the number of bytes required by the user's program text and common variables carried through the sort.

Coding of the input procedure must conform to the following rules:

1. The SORT-4 variables listed in Appendix B must be treated as Read Only variables; that is, their values must not be altered by the user's program. Any variable employed in a special input procedure for purposes other than reading must <u>not</u> be one of the reserved SORT-4 variables listed in Appendix B.

2. 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.

3. 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 4-2.

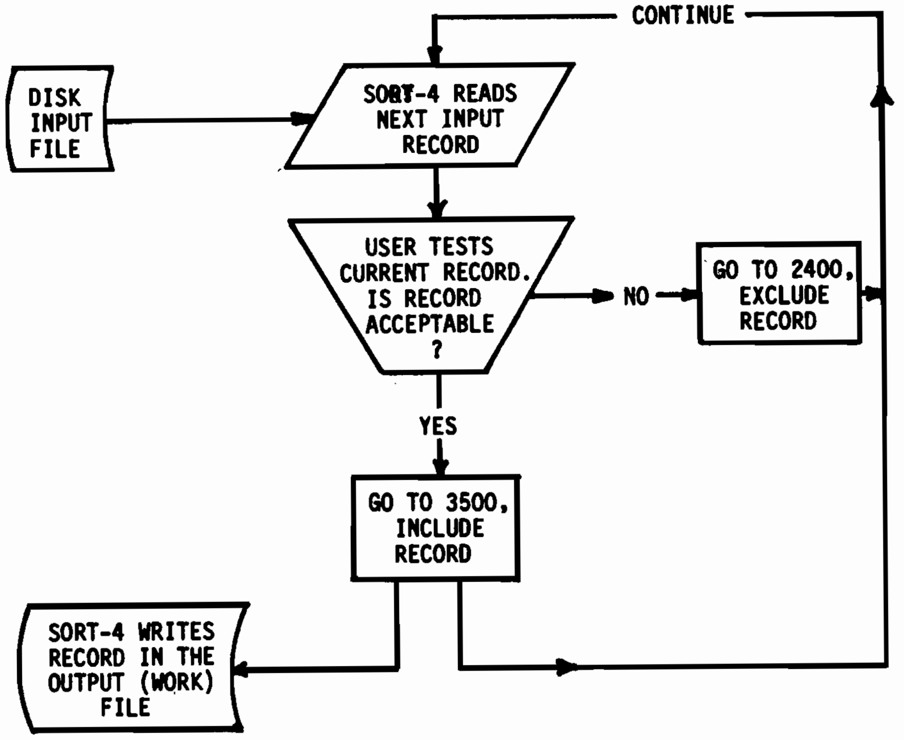

CONTINUE
DISK INPUT FILE
SORT-4 READS NEXT INPUT RECORD
USER TESTS CURRENT RECORD. IS RECORD ACCEPTABLE ?
NO
GO TO 2400, EXCLUDE RECORD
YES
GO TO 3500, INCLUDE RECORD
SORT-4 WRITES RECORD IN THE OUTPUT (WORK) FILE

Figure 4-2. 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 SF'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 setup 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 setup 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$=blank or "A"), SORT-4 adds another dimension for blocking to the variables to which the blocking applies. With array-type blocking, the extra dimension is added to read fields. With packed array format, the extra dimension is added to packed fields. In the example above, where records were written five per block, the variable names would become A0(5), A1$(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 two 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.

## 4.15 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 4.7) 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$8, etc. The value of S9 in the setup module indicates how many of these common variables (0, 1, or 2) will be saved with the appropriate information, when loading user program M$. The record count and error code (if any) might be displayed by the user program loaded by SORT-4, although SORT-4 stops execution if an error occurs.

## 4.16 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 (setup 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 setup 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 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 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 4.17, below.

If error messages are not passed to the user program, any error message encountered will be displayed, as listed in Table 4-4. Otherwise, upon successful completion, the following is displayed:

```
RECORD COUNT NNNNN
STOP END OF SORT
```

## 4.17 ERROR MESSAGES AND RECOVERY PROCEDURES

There are two types of error conditions that could be encountered during the operation of SORT-4. Hardware (ERR lnn) errors 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."

### Hardware Errors

Certain hardware error messages will cause SORT-4 to stop. If any of these occur, touch SF'31 to end the sort and close any files that may be open in a multistation environment.

If either ERR I96 or ERR I99 occur, touch SF'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.

ERR D83 indicates that the output file to be cataloged is already cataloged. Touch SF'31 to end the program. Make sure that the correct disks are mounted. This may require a programming change in the setup module (C$ = "Y").

ERR D82 indicates the file cannot be found. A data file or program module does not exist on the specified device. Touch SF'31 to end the program. Make sure that the correct disks are mounted. This may require a programming change in the setup module.

Other hardware errors could be caused by hardware or software failure, invalid data, or a number of other 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 SF'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. Associated with each error condition is a message and a number. The number is optionally passed to a user program via S9 if the option is specified; otherwise, the accompanying message is displayed on the screen and SORT-4 stops. (Also see Section 4.15.)

Error messages displayed by SORT-4 are listed in alphabetic order in Table 4-4, with a cross-reference to the number associated with each error message. Error messages and recovery procedures are listed in numeric order following the table.

Table 4-4. Alphabetic List of SORT-4 Error Messages

| ERROR MESSAGE | NUMBER* |
|---|---|
| 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 |

* For an explanation of the error message and its recommended recovery procedure, refer to the appropriate number on pages 125-131.

## 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 an END control sector (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 setup 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 setup program (file name, password).

4. INVALID RECORDS PER BLOCK

   Blocking (B) is not an integer from 1 to 255, or does not match actual blocking in sample record, or does not match blocking (V8$) in KFAM KDR record.

   Recovery: Correct setup program.

5. INVALID STARTING BLOCK

   Starting block # to be sorted (D) is less than 1 or not an integer.

   Recovery: Correct setup 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 setup 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 setup program should be corrected.

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 for another file. The device address must be consistent (multistation or not multistation) for all files on that device.

Recovery: Correct the setup program.

10. INVALID DEVICE ADDRESS

A device address is specified which is either blank or not in the table of device addresses, MO$().

Recovery: Correct the setup 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 setup program.

Recovery: Mount correct disk and rerun, or correct the setup 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.

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 setup 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 setup program (file name, password, device).

16. WORK FILE TOO SMALL

The work file contains less than 25 sectors, or is too small to either sort the maximum number of records specified (P8) or 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 setup 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 setup 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.

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 setup 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, 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 setup 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 setup 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$() is blank, or the sample record shows more than one read field array, or the array is numeric. This error is also possibly caused by the definition in A$() not fitting the sample record: too many or too few fields are defined; field lengths of packed fields do not add up to field lengths of read fields; or field lengths, types, and array dimensions do not match the sample record.

Recovery: Correct the setup 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 setup program.

24. INVALID RECORD TYPE (F$)

The record format (F$) is not blank, "A", "P", "T", or "V".

Recovery: Correct the setup 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 setup 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 setup program.

27. SORT KEY TOO LONG

The total length of the sort key exceeds 64 bytes.

Recovery: Change the setup program to shorten the sort key, if possible.

28. OPERATOR INTERRUPT

SORT-4 was terminated by depressing SF'31.

Recovery: Rerun the program.

29. NO CPU NUMBER

No station (CPU) number or an invalid station number was specified with multistation files.

Recovery: Specify the station number (S2) in the setup 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 setup program.

31. INVALID SORT TYPE

The type of sort specified (P8$) is not blank, "K", "R", or "T".

Recovery: Correct the setup 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 setup 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 five records per block and occupy at least two 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 setup 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 control sector on the output file or reopening the work file used for output, or closing a file.

Recovery: Rerun the program. Notify Wang Laboratories if this error recurs.

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 setup 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 two or more elements.

Recovery: Change F$ to blank in the setup program, or change the format of the input record on disk so that at least two elements of an array are written per block of packed records.

39. INPUT FILE BUSY

Multistation input file cannot be opened in the Read Only access 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 non-KFAM file was used as input, or see the KFAM utility Reset Access Table (either KFAM-3, -4, -5, or -7), if a KFAM input file was specified.

40. WORK FILE BUSY

Multistation work file cannot be opened in the Exclusive access mode because of an access mode 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.

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.

## 4.18 SORT-4 TIMINGS

In general, SORT-4 takes longer for the setup 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 setup time, SORT-4 will run faster than SORT-3. Table 4-5 gives some comparisons of running times for different SORT-4 machine configurations and input files. An explanation of the terms used in the table follows:

- 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, blocked two per sector, with an 8-byte (numeric) sort key. The record contains six fields.
- File 120/64 is the same file as above, except that four fields (one numeric and three alpha) or a total of 64 bytes, are included in the sort key.
- Sort type R is a full record sort, K is a key sort, and T is a tag sort.

Input records are in random order. A five-megabyte hard disk is used in all cases. All sort times are in minutes, and vary depending on the number of fields in a record, the type of sort, the amount of memory available, and other factors.

Table 4-5. SORT-4 Timings, 2200VP

| FILE | RECORDS | SORT TYPE | SORT-4 TIME 2200VP w/16K | SORT-4 TIME 2200VP w/ 64K |
|---|---|---|---|---|
| 24/24 | 2000 | R | 1.0 | n/a |
| 24/24 | 20000 | R | 9.1 | 6.8 |
| 120/8 | 4000 | K | 9.2 | 9.0 |
| 120/8 | 4000 | R | 9.55 | 6.9 |
| 120/8 | 4000 | T | 2.5 | 2.5 |
| 120/64 | 1000 | K | 3.0 | n/a |
| 120/64 | 1000 | R | 2.3 | 1.8 |
| 120/64 | 4000 | R | n/a | 7.0 |

## 4.19 SAMPLE SETUP PROGRAMS

Table 4-1 provides a master setup program that includes all setup statements that could possibly be needed. As an aid to first-time SORT-4 users, the following sample setup programs are provided.

## Sorting a KFAM-3 Data File

A list of publications is contained in a KFAM-3 data file, where each record is composed of the following fields:

1. A 12-byte alphanumeric publication number; e.g., 700-5010B, is the KFAM-3 key field.

2. A 2-byte alphanumeric flag, indicating whether the document is obsolete or still active.

3. A 60-byte alphanumeric title and description of the document is the sort key field; in this case, it composes the entire sort key.

The records are blocked three per sector in array format and are not packed. The User File name is PUBLF000, and the Key File number is 1. The purpose of the sort is to create an output file sorted by the title/description field, the third field in each record. Since this list is updated biweekly, the input User File/Key File resides on diskette at address B10. The work file resides on removable disk at address B20, and the output file resides on diskette at address 310. A 2200VP central processor is used.

The setup program first provides the memory size, disk addresses, and SELECT statements to define the hardware configuration. Other statements are documented by REM statements and are explained elsewhere in this chapter, especially in Table 4-1. Note that the following statement lines (see Table 4-1) were not included, for the following reasons:

1. Line 3430, input file password, is unnecessary since KFAM-3 files do not have file passwords, and multistation operation (hog mode addresses) has not been specified.

2. Line 3450, DC/DA or BA mode, is unnecessary since the default value of blank (DC/DA mode) is acceptable.

3. Line 3455, record format, is unnecessary since the default value of blank (not packed) is acceptable.

4. Line 3460, length of the variable portion of the record, is unnecessary since the file contains only fixed-length records, and the default value of 0 (zero) is acceptable.

5. Line 3465, description of packed fields, is unnecessary since there are no packed fields (the record format is blank).

6. Line 3475 and 3480 are unnecessary since the file is a KFAM file, not a sequential file.

7. Line 3480 and 3485 are unnecessary since the entire KFAM is to be sorted.

8. Line 3530 is unnecessary since the sort work file has no password, and multistation operation has not been specified.

9. Line 3540 is unnecessary since the sort work file is already cataloged.

10. Line 3560 is unnecessary since the output file has no password, and multistation operation has not been specified.

11. Line 3565 is unnecessary since the default value of Y (output file is cataloged) is acceptable.

12. Line 3580 is unnecessary since deferred mounting is not needed, thereby making the default value (blank) acceptable.

13. Line 3585 is unnecessary since multistation operation has not been specified for any disk device.

14. Lines 3595 and 3600 are unnecessary since a special input procedure is not used.

The SORT-4 setup program for this application appears on the following page.

```
10 REM "TITLE" SORTS A KFAM-3 DATA FILE WITH BLOCKED DATA RECORDS
20      DIM K(10), B(10), N(10), N$(4)8, P$(4)16, F$(6)3, A$(4)62, MO$(1)21
179     LOAD DC F"SORT4" 10,179
3400    M=SPACEK: REM 2200VP-RESIDENT PROGRAM
3404 REM DEFINE DISK DEVICES IN THE CONFIGURATION
3405    MO$(1)="310B10B20"
3410    SELECT DISK 310: SELECT #1/B10: SELECT #2/B20
3414 REM DEFINE KFAM-3 FILE, FILE TYPE 3 (F), INPUT FILE NAME AND ADDRESS
3415    F=3
3420    N$(1)="PUBLF000"
3425    F$(1)="B10"
3434 REM DEFINE KEY FILE NUMBER (J) AND ADDRESS
3435    J=1
3440    F$(2)="B10"
3444 REM DEFINE RECORD BLOCKING (B)
3445    B=3
3474 REM DEFINE SORT KEY FIELD
3475    D=1
3495    K=1
3500    K(1)=3: REM THIRD FIELD IS THE SORT KEY FIELD
3519 REM DEFINE WORK FILE NAME AND ADDRESS
3520    N$(3)="SORTWORK"
3525    F$(3)="B20"
3549 REM DEFINE OUTPUT FILE NAME, ADDRESS, AND ITS SIZE
3550    N$(4)="SORTOUT"
3555    F$(4)="310"
3570    P7=680
3574 REM FORCE FULL-RECORD SORT
3575    P8$="R"
3589 REM SORT-4 PROGRAMS LOADING ADDRESS
3590    F$(5)="310"
3604 REM NAME AND ADDRESS OF PROGRAM TO LOAD FOLLOWING COMPLETION OF SORT-4
3605    M$="TITLE2"
3610    S9=2: REM PASS RECORD COUNT AND ERROR CODE TO "TITLE2"
3615    F$(6)="310"
```

Figure 4-3. The SORT-4 Setup Program

## Sorting an Inventory File

A file of inventory records is stored in a sequential file. Before each record is written to disk, it is packed into a single alphanumeric array using the field form of the $PACK statement. Table 4-6 shows the individual fields within the packed record, and the following program text defines each record's contents as stored one record per sector.

Table 4-6. Inventory Record Layout

| FIELD NUMBER | VARIABLE NAME | PACKED BYTE LENGTH | BYTE LOCATION | FIELD TYPE |
|---|---|---|---|---|
| 1 | F1$ | 12 | 1-12 | Alphanumeric |
| 2 | F$ | 12 | 13-24 | Alphanumeric |
| 3 | F2$ | 24 | 25-48 | Alphanumeric |
| 4 | F3$ | 2 | 49-50 | Alphanumeric |
| 5 | H0 | 3 | 51-53 | Numeric, IBM Packed Decimal |
| 6 | G2$ | 5 | 54-58 | Alphanumeric |
| 7 | G3$ | 24 | 59-82 | Alphanumeric |
| 8 | I5$ | 6 | 83-88 | Alphanumeric |
| 9 | I4$ | 20 | 89-108 | Alphanumeric |
| 10 | E | 4 | 109-112 | Numeric, IBM Packed Decimal |
| 11 | E1 | 4 | 113-116 | Numeric, IBM Packed Decimal |
| 12 | E2 | 4 | 117-120 | Numeric, IBM Packed Decimal |
| 13 | E3 | 4 | 121-124 | Numeric, IBM Packed Decimal |
| 14 | E4 | 4 | 125-128 | Numeric, IBM Packed Decimal |
| 15 | E5 | 4 | 129-132 | Numeric, IBM Packed Decimal |
| 16 | H | 4 | 133-136 | Numeric, IBM Packed Decimal |
| 17 | H2 | 4 | 137-140 | Numeric, IBM Packed Decimal |

The sort key is comprised of two fields. The contents of alpha-variable I5$ (field number 8) and alpha-variable F1$ (field number 1) are chosen as the primary and secondary sort key fields, respectively. If the record type is defined as packed (F$="P"), the alpha-array A$() must be provided to define the packed fields and the sort key is simply defined by the relative field numbers, 8 and 1.

Alternatively, each record may be viewed as being comprised of the elements of alpha-array M$() and the sort key fields specified as partial fields of each array element. The record type is specified as not packed (F$="blank") and the alpha-array A$() need not be provided in the setup program.

To illustrate the two approaches to defining the record type and sort key fields, two setup programs which perform the same task are provided: a 2200MVP and a single Model 2280 disk surface (disk address D13) are used. A tag sort is performed, and the program INVTPRNT is loaded following completion of SORT-4 operation, which prints the input file records in sorted order by using the pointers in the work file. Note that Hog mode addresses are used when defining the disk device addresses, array elements of alpha-array F$(), since the input file is a multistation/multiplexed file.

In both setup programs, the following parameters were not defined because their values are not required:

1. Key File number (J) on line 3435.
2. Key File address (F$(2)) on line 3440.
3. Length of variable portion of record (N6) on line 3460.
4. The starting and ending key values to be sorted (A$,E$) on lines 3485 and 3490.
5. The maximum number of records to be sorted (P8) on line 3540.
6. The number of sectors in the output file (P7) on line 3570.
7. The deferred mounting option (D$) on line 3580.
8. The name of the special input procedure and its number of bytes (G$ and M4) on lines 3595 and 3600.
9. The error recovery method (S9) on line 3610.

The following parameters are not included (in both programs) since their default values are acceptable:

1. Input file password (P$(1)) on line 3430.
2. Records per block (B) on line 3445.
3. DC/DA or BA mode records (B$) on line 3450.
4. Starting number and number of blocks (D and L$) on lines 3475 and 3480.
5. The sort work file and output file passwords (P$(3), P$(4)) on lines 3530 and 3560.

Note that the sort key is specified differently in the two setup programs. The program INVTSORT specifies that records are not packed (F$="blank") by means of exclusion of this parameter, and uses partial fields (array elements) to specify the two sort key field locations. The program SORTINVT specifies packed records (F$="P"), provides the alpha-array A$() to define the fields, and defines the partial sort key fields in terms of array M$().

```
10 REM INVTSORT, SORT USING PARTIAL FIELDS
20      DIM K(10),N(10),B(10),F$(6)3,N$(4)8,MO$(1)21
179     LOAD T"SORT4"10,179
3400      M=SPACEK+2       : REM MEMORY SIZE
3405      MO$()="D10D11D12D13D14D15"
3410      FOR I=0TO 5
3412         SELECT #I<STR(MO$(),I*3+1,3)>
3414      NEXT I
3415      F=0                 : REM FILE TYPE
3420      N$(1)="INVTF010"    : REM INPUT FILE NAME
3430      F$(1)="D93"
3495      K=2                 : REM NUMBER OF KEY FIELDS
3505  REM SORT KEY SPECIFICATIONS
3515      K(1)=2
3525      K(2)=1
3535      B(1)=21
3545      B(2)=1
3555      N(1)=6
3565      N(2)=12
3575      N$(3),N$(4)="SORTWORK"
3585      F$(3),F$(4)="D93"
3590      C$="W"              : REM WRITE TAGS INTO SORTWORK FILE
3595      P8$="T"             : REM TYPE OF SORT (TAG)
3600      S2=1
3605      M$="INVTPRNT"       : REM PROGRAM TO LOAD FOLLOWING SORT
3625      F$(5),F$(6)="D93"
```

Figure 4-4. The INVTSORT Program

```
10 REM SORTINVT, SORT USING PACKED RECORD DEFINITION
20      DIM K(10),N(10),B(10),F$(6)3,N$(4)8,MO$(1)21,A$(4)62
179     LOAD T"SORT4"10,179
3400      M=SPACEK+2      : REM MEMORY SIZE
3405      MO$()="D10D11D12D13D14D15"
3410      FOR I=OTO 5
3412         SELECT #I<STR(MO$(),I*3+1,3)>
3414      NEXT I
3415      F=0                : REM FILE TYPE
3420      N$(1)="INVTF010"   : REM INPUT FILE NAME
3425      F$(1)="D93"
3500      K=2                : REM NUMBER OF KEY FIELDS
3510   REM SORT KEY SPECIFICATIONS
3520      F$="P"
3530      A$()="A12,A12,A24,A2,P3,A5,A24,A6,P4,P4,P4,P4,P4,P4,P4,P4,A20"
3540      K(1)=8
3550      K(2)=1
3560      N$(3),N$(4)="SORTWORK"
3570      F$(3),F$(4)="D93"
3575      C$="W"             : REM WRITE TAGS INTO SORTWORK FILE
3580      P8$="T"            : REM TYPE OF SORT (TAG)
3585      S2=1              : REM STATION NUMBER FOR MULTISTATION/MULTIPLEXED
3590      M$="INVTPRNT"      : REM PROGRAM TO LOAD FOLLOWING SORT
3600      F$(5),F$(6)="D93"
```

Figure 4-5. The SORTINVT Program

# CHAPTER 5
# KFAM-7 GENERAL INFORMATION

## 5.1 KFAM AND DISK ACCESS METHODS

A disk access method transfers 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 rapidly rotating disk(ette) surface by using a moveable read/write head. 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 (referred to as KFAM) provides several means of rapid access to records within a file.

KFAM, although unique, resembles access methods usually categorized as "indexed sequential" or "indexed." A KFAM file consists of two files: (1) the file containing the data records, called a User File, and (2) a file containing an "index" for quickly locating specific User File records, called a Key File. Within each User File data record is a "key," such as a social security number. Each key's value is typically unique within the same KFAM 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 File, which enables the 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:

- An entry in the Key File is maintained for each record in the User File, allowing records to be added to the KFAM file in random order of their keys and, thereafter, accessed either by key or key sequence. Record deletions by key are also provided. Disk space is efficiently utilized regardless of logical record length since KFAM fully supports blocked records (multiple records per sector), single sector records (not blocked), and multiple sector records.

- Records entered in random key order may be reordered by supplied KFAM utility software into ascending order of their keys. (With multiple Key Files, reorganization is based upon the "primary" Key File.) This provides efficient record access by ascending or descending key sequence.

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

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

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

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

- Three versions of KFAM-7 are provided which support multiple Key Files for a single User File, allowing alternate (secondary) key retrieval and multiple key paths for User File records. One version requires special programming considerations to prevent duplicate updating of records (sector protection is not available).

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

The disk catalog system allows files to be easily found on a specified disk, but does not keep track of individual records within a file. KFAM is a system for both keeping track of, and rapidly locating, individual records within a file.

## The Index of Data Records

Within the Key File, KFAM creates and maintains an index of the individual User File records. Each record is uniquely identified by a key field. The index can be thought of as a list of all the keys for a given file. Associated with each key in the index is the relative sector address of the record that the key identifies.

The Key File index constructed by KFAM is a sophisticated B-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. Keys can be added and deleted easily, without disturbing the organization of the Key File.

Different Key File operations are performed by different KFAM subroutines. The Open subroutine initializes certain memory-resident KFAM variables with information read from the Key File and performs the necessary 2200 disk operations to provide access to the User File and the Key File. Conversely, the Close subroutine rewrites certain KFAM variables into the Key File to reflect its updated contents and performs the necessary 2200 disk operations to discontinue access to the User File and the Key File. During the execution of other KFAM subroutines, Key File information may be rewritten automatically when it is necessary to reflect changes to its contents, depending upon the KFAM-7 version in use and the access mode chosen when opening the file.

KFAM subroutines do all of the searching and updating within the Key File. KFAM subroutines are provided to add or delete a key from the Key File, access records based up a supplied key or key sequence (ascending or descendinơ), and perform other functions. Through the use of KFAM subroutines, KFAM coordinates multiuser file access by using file password protection and four available access modes; it also coordinates multiuser record access within a file including record protection (for two of the three KFAM-7 versions). Most of the Key File maintenance is automatically handled by KFAM during subroutine execution.

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 statement (DC mode) can be executed, at the desired KFAM-controlled sector location. A KFAM variable containing the relative sector address of the appropriate record is available for use with DATALOAD DA or DATASAVE DA statements (DA mode) and DATALOAD BA or DATASAVE BA statements (BA mode).

KFAM-7 is a multiuser system allowing up to 16 stations to simultaneously access a KFAM User File. KFAM-7 employs protective procedures to prevent the destructive instrusions of one station into the operation of another and offers levels of protection consistent with the required operations, thereby allowing maximum availability of the file and the disk surface to other stations. The levels of protection are provided by four available access modes, where certain conventions are associated with each access mode. Each station chooses an access mode when requesting access to (opening) a file. The access modes are the following:

1. "Inquiry" access mode allows a station granted access to only read within the User File; other stations may both read or write.

2. "Read Only" access mode allows a station granted access to only read within the User File; other stations may likewise only read.

3. "Shared" access mode allows a station granted access to both read and write within the User File; other stations may likewise read or write.

4. "Exclusive" access mode allows only one station to access the User File; that station may read or write within the file once access is granted.

A record protection option is available for Shared access mode operation, whereby only the station requesting record protection has access to that record until another KFAM subroutine is executed by that station. Record protection is typically used while updating a record, and is like having exclusive access to the protected record.

## 5.2 FILE STRUCTURES

For each KFAM file, KFAM creates and maintains certain information stored in the User File and the Key File. The User File contains data records maintained by the user. Certain sectors in the User File, however, also contain specific information maintained by KFAM. In contrast, the entire contents of the Key File are maintained by KFAM.

### User File Structure and Maintaining File/Record Access Information

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 subroutines and DATASAVE statements. An END control sector (end-of-data) follows the last sector of data. The position of the END control sector is automatically controlled by KFAM, based upon the KFAM subroutines called by the user's program. One or more extra sectors may follow the END control sector. In the next-to-last sector, KFAM maintains a dummy END control sector which, at the programmer's option, also contains recovery information necessary to recreate the Key File should it be accidentally destroyed (which consists of most of the Key File's Key Directory Record (KDR) and a list of the last record position assigned within a sector by each station number). The last sector allocated is the catalog trailer control sector.

Access to the Key File is determined by access to the User File. The User File's catalog trailer control sector contains an access table which is examined before the User File is opened and determines whether User File access is granted or refused, based upon 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. If a User File has a password that password must be supplied to access the User File and Key File.

Note that the two END control sectors and the catalog trailer control sector are indicated by the first two bytes which contain specific hexadecimal values. Figure 5-1 illustrates the User File structure.

The END control sector (end-of-data) and the dummy END control sector (recovery information) are usually 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 control sector allows the file to be read sequentially, provided the user observes the following conventions:

1. All deleted records must be flagged with a hexadecimal value of FF; i.e., HEX(FF), in the first byte of the key.

2. With blocked records (types A, B, and C), whenever a new sector is allocated, all records in the block must be initialized with a HEX(FF) in the first byte of the key.

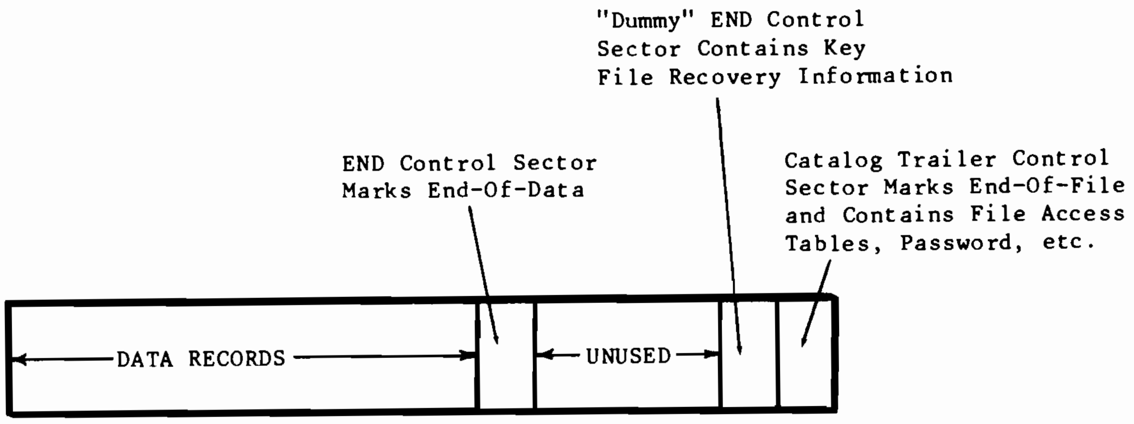

"Dummy" END Control Sector Contains Key File Recovery Information
END Control Sector Marks End-Of-Data
Catalog Trailer Control Sector Marks End-Of-File and Contains File Access Tables, Password, etc.
DATA RECORDS
UNUSED

Figure 5-1. User File Structure

## Key File Structure

The Key File is the means by which KFAM facilitates indexed record access to randomly dispersed records within the User File. The first Key File sector contains the Key Directory Record (KDR), and each remaining sector contains a Key Index Record (KIR). The Key File also contains an END control sector which is automatically rewritten as sectors are added to the Key File. A catalog trailer control sector is the last sector allocated to the Key File.

The KDR control information maintains each station's current sector address, any sectors protected by a station, information about the KFAM file, and system (KFAM internal) information. With certain versions of KFAM-7 provided in ISS-5, record access information is dynamically maintained in memory instead of in the KDR. The KDR is rewritten under certain conditions with all KFAM-7 versions 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 control sector, consists of nearly all of the KDR.

Each remaining Key File sector contains a Key Index Record (KIR), which consists of Key Index Entries (KIEs). KIEs are internal tables consisting of key values and corresponding record locations, by which KFAM provides indexed access to each User File record. When the programmer adds keys to a Key File, an adjustable bias is available to maximize the number of KIEs per KIR, thus reducing the time needed for KFAM to search the Key File for a key.

The KDR is rewritten when a file is closed and when a FINDNEW, FINDNEW(HERE), or DELETE subroutine is called. With the Multiplexed KFAM-7 version, the KDR is also rewritten at least once during each successful record access subroutine call in access modes where multiuser record access with updating is allowed (Inquiry and Shared access modes). This preserves Key File integrity by ensuring that only one subroutine executes at any time.

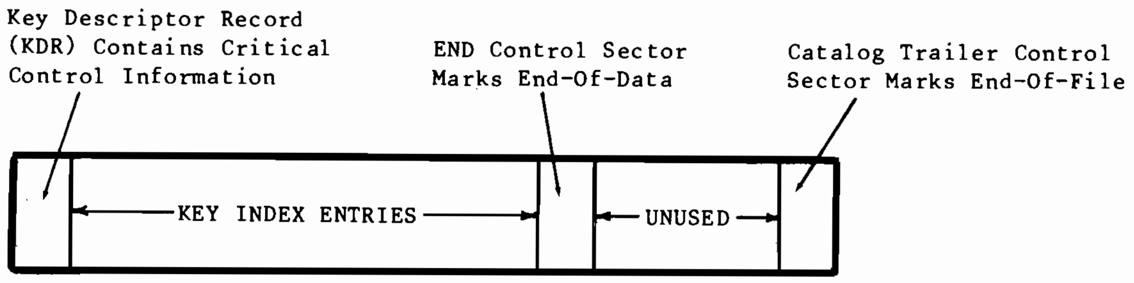


Figure 5-2. Key File Structure

## 5.3 SOFTWARE COMPONENTS

KFAM-7 software falls into the two major groups of utilities and subroutines. KFAM-7 utilities are available to initialize a User File and Key File, reorganize a User File and Key File, print the contents of the Key File, recover from certain operational accidents, and perform other functions. KFAM-7 subroutines are marked (DEFFN' statement) subroutines which open and close KFAM files, locate User File records, and add or delete keys in the Key File. KFAM-7 subroutines are the operational heart of the Key File Access Method.

### KFAM Utilities

Following ISS start-up procedures, KFAM utility programs may be chosen in reply to the KFAM-7 menu (exception: Reorganize/Rebuild Subsystem). ISS start-up procedures maintain certain information, including valid disk device addresses and the device address of a printer. This information is available through the use of common variables which, for instance, are used by KFAM utilities to ensure that a disk address entered in reply to an operator prompt is indeed a valid disk address. The KFAM-7 utilities are the following:

INITIALIZE KFAM FILE -- This setup utility catalogs (creates) a User File and/or a Key File and writes the KDR into the Key File based upon certain information entered by the user. This is always the first step in creating any KFAM User File/Key File.

BUILD KEY FILE -- After running Initialize KFAM File, this setup utility is run if the User File contains any data records to create a key/relative sector address entry (KIE) in the Key File for each data record in the User File. The KDR information is also updated.

PRINT KEY FILE -- This utility prints or displays the KDR, each KIR, and all KIEs in each KIR. Certain access table information is output, which provides the access mode of each possible station (1 through 16) accessing that file. This information is not only helpful in determining the cause of access mode conflicts, but also allows the Key File structure to be studied.

REALLOCATE FILE SPACE -- This special-purpose utility updates internal KDR information related to the size of a User File and Key File. The ISS utility called Copy/Verify may be used to copy a User File and/or a Key File and increase or decrease the number of extra sectors in each file; however, the Copy/Verify utility does not update critical KDR information. Reallocate File Space is required to update this KDR information following Copy/Verify operation.

REORGANIZE IN PLACE -- This utility program reorders each record in the User File into ascending key sequence, removes all deleted records from the User File, and then rebuilds the Key File based upon the reordered User File. Following reorganization, the time needed to access the file in ascending or descending key sequence is reduced. This utility rewrites the reordered User File and Key File over the original (input) User File and Key File.

REORGANIZE/REBUILD SUBSYSTEM -- This program-controlled subsystem reorganizes a User File and Key File like the Reorganize In Place utility, but writes the reordered User File and Key File into separate output files without overwriting the original input User File and Key File. A short user-written program defines the reorganization parameters. This subsystem can be used to copy a User File and Key File since it leaves the input User File and Key File intact, and can copy, reorganize, and change file space allocation at the same time. This subsystem allows a Key File to be built without reorganizing the input User File, and is a program-controlled alternative to the Build Key File utility.

CONVERT TO KFAM-7 -- This utility accepts a KFAM-3 or -4 data file as input and converts it to KFAM-7 format. KFAM-5 files are media compatible with KFAM-7 files and do not require conversion.

BUILD SUBROUTINE MODULE -- A complete set of KFAM-7 subroutines reside on the supplied KFAM-7 diskette for each KFAM-7 version. This utility allows only those subroutines needed for a particular application to be chosen, and saves them into a program file. When loaded, the subset of subroutines requires less memory than the complete set of subroutines.

KEY FILE RECOVERY -- This recovery utility is provided to reconstruct an accidentally destroyed Key File. The User File must be intact for this program to operate successfully. This utility is similar to the Build Key File utility, but uses the recovery information stored in the next-to-last User File sector to recreate the Key File's KDR.

RESET ACCESS TABLES -- KFAM-7 maintains control information in various access tables. This information may be erroneous if the file is not closed due to a power failure or operational accident. This utility is provided to clear the erroneous access table information; the Print Key File utility is used to determine that the information is erroneous. The access table information may be cleared for a particular station number or all possible stations; it may be cleared for specified files or all files listed as open in global memory.

## KFAM Subroutines

KFAM-7 subroutines are available in three versions to accommodate different hardware arrangements. Whether the KFAM-7 subroutines reside in a memory partition self-designated as "global" in a Single or Multiple Bank 2200MVP central processor, or are coresident with user-written program text in a 2200VP central processor, the available subroutines are the same and the programming procedures are nearly identical.

Subroutines fall into the following groups: General Purpose, Key Sequence Access, Random Access, Add and Delete, and Special Purpose. Table 5-1 lists and describes each subroutine.

Table 5-1. KFAM-7 Subroutines

| GROUP | NAME | DESCRIPTION |
|---|---|---|
| General Purpose | OPEN | Opens the specified User File and Key File for this station. KFAM variables necessary for record access are initialized. |
| | CLOSE | Closes the specified User File and Key File for this station, and rewrites the KDR in the Key File and the END control sector and recovery information in the User File. KFAM variables no longer needed for record access are reset. |
| | RE-OPEN | Changes the access mode of a currently open KFAM file. |
| | WRITE RECOVERY INFORMATION | Rewrites the END control sector and recovery information in the User File without closing the file. |
| Key Sequence Access | FINDFIRST | Locates the lowest key in the Key File and sets the User File Current Sector Address to the corresponding record location. |
| | FINDNEXT | Locates the next-highest key in the Key File and sets the User File Current Sector Address to the corresponding record location. After FINDFIRST finds the lowest key, repeated FINDNEXT subroutine calls allow ascending key sequence access throughout the entire file, since each call locates the next record in logical key sequence. FINDNEXT can be preceded by any successful subroutine call in the Key Sequence Access, Random Access, and Add and Delete groups to return the next record in ascending key sequence. |
| | FINDLAST | Locates the highest key in the Key File and sets the User File Current Sector Address to the corresponding record location. |

Table 5-1. KFAM-7 Subroutines (continued)

| GROUP | NAME | DESCRIPTION |
|---|---|---|
| | FINDPREVIOUS | Locates the next-lowest key in the Key File and sets the User File Current Sector Address to the corresponding record location. After FINDLAST finds the highest key, repeated FINDPREVIOUS subroutine calls allow descending key sequence access throughout the entire file, since each call locates the previous record in logical key sequence. FINDPREVIOUS can be preceded by any successful subroutine call in the Key Sequence Access, Random Access, and Add and Delete groups to return the previous record in descending key sequence. |
| Random Access | FINDOLD | Locates the supplied key in the Key File and sets the User File Current Sector Address to the record with that key. |
| Add and Delete | FINDNEW | Adds the supplied key to the Key File, allocates a User File location for the new record, and sets the User File Current Sector Address to that record location. The number of active records increases by one, the assigned record location updates the current "last record" control information, and the KDR is rewritten. |
| | FINDNEW(HERE) | Adds the supplied key to the Key File and sets the internal KIE relative sector address to the current User File sector. This subroutine typically follows a DELETE subroutine call and either changes the value of a key in an existing record or allows the disk space used by a deleted User File record to be reused without reorganization. The number of active records increases by one, the current "last record" control information is not updated, and the KDR is rewritten. |
| | DELETE | Removes the specified key from the Key File and sets the User File Current Sector Address to the record whose key is deleted. The number of active records decreases by one, and the KDR is rewritten. |

Table 5-1. KFAM-7 Subroutines (continued)

| GROUP | NAME | DESCRIPTION |
|---|---|---|
| Special Purpose | RELEASE | "Unprotects" the previously protected User File record. Each station may flag a record to be updated during a record access subroutine call; however, no other stations may access the protected record until that station calls another KFAM subroutine. Should a long delay occur, RELEASE may be called to allow other stations to access the protected record. |

## 5.4 KFAM, THE 2200 DEVICE TABLE, AND 2200 DISK STORAGE CHARACTERISTICS

KFAM-7 performs two very important tasks for the user: (1) it maintains an index for all data records in a User File, and (2) it controls file and record access by multiple users. This is accomplished internally by KFAM-7 as a result of KFAM-7 subroutine use by application programs. The user operates indirectly on the Key File via KFAM subroutine calls. KFAM interprets each subroutine call and executes the requested operation only if it is considered valid. Other precautions are also performed by KFAM to preserve Key File integrity, although unseen by the user.

Unlike the Key File which the programmer only indirectly accesses, the User File is the user's complete responsibility to maintain in accordance with KFAM-7 requirements and conventions. The user must ensure that User File records and their respective keys are in agreement with the key value/sector address entries in the Key File and other Key File information. In most cases, calling the appropriate KFAM subroutines, careful use of the BASIC-2 language DATASAVE and DATALOAD statements, and other considerations are necessary to avoid programming accidents.

### The 2200 Device Table

Each 2200VP central processor or 2200MVP memory partition (station) contains its own device table in memory. Each device table maintains certain information including 16 slots, each of which correspond to one disk file (or non-disk device). A slot contains control information including the file's disk device address, the absolute sector address where the file begins, the absolute sector address at which this station is currently positioned in that file, and the absolute sector address where the file ends.

BASIC-2 syntax requires that a pound sign (#) precede each reference to a slot number, usually called a "file number." The Open subroutine requires certain parameters to be included in its argument list. These parameters include the disk device address and the file number to be associated with the User File, and the disk device address and the file number to be associated with the Key File. The Open subroutine executes two SELECT statements which place the disk addresses into the corresponding device table slots, or file numbers, specified for the User File and Key File. The disk operations associated with opening 2200 disk files are executed for both files, which places the beginning, current, and ending sector locations into the corresponding slots in the device table for both files (obtained from the disk index).

Other Open subroutine arguments include the name of the User File, a KFAM ID number from one to eight by which this pair of files is later identified, and a Key File number from which KFAM creates the Key File name. The KFAM ID number is used internally by KFAM and is also used in user-specified KFAM subroutine argument lists for record access operations.

After a KFAM file has been opened, data records can be written into, and/or read from, the User File. The user must be extremely careful <u>not</u> to write any data into the Key File. Given the KFAM ID number as an argument, a particular KFAM subroutine call results in a specific action, during which the Key File is usually accessed internally by KFAM.

A call to a KFAM record access subroutine might result in KFAM (1) locating the desired key in the Key File and (2) placing the User File relative sector address obtained from the Key File into the Current Sector Address in the device table slot for the User File. KFAM sets the Current Sector Address by performing a DBACKSPACE BEG statement, followed by a DSKIP statement to the value of variable T6 (relative sector address) which it obtains from the Key File index. The user can then access the desired User File record knowing the record is located at the Current Sector Address for that station.

The 2200 system relies upon device table information to read or write a data record. The DATALOAD DC statement loads a data record based upon the specified file (slot) number's Current Sector Address. Similarly, the DATASAVE DC statement saves a data record based upon the specified file number's Current Sector Address. With DATASAVE BA or DA and DATALOAD BA or DA statements, which do not use the device table Current Sector Address, the user program can specify the correct absolute sector address by adding the relative sector address returned in KFAM variable T6 to the absolute sector address at which the file begins obtained by using a LIMITS statement. (See the <u>Wang BASIC-2 Disk Reference Manual</u>.)

In both cases, each DATALOAD or DATASAVE statement specifies (1) the file number within which a record is to be read or written, and (2) an argument list of variables which either contain the data to be written (DATASAVE statement), or will contain the data to be read (DATALOAD statement). The argument list of variables determines the record length, which must be uniform for all records within the same KFAM User File. More specifically, it is strongly recommended that the argument list variables used to write records within the file be the same; that is, the sequence, dimensioned lengths, and the type of variable (alpha or numeric, scalar or array) should be identical for each argument list. The same argument list is also required for reading records.

## 2200 Disk Storage

The 2200 system writes a logical data record comprised of the data contained in the argument list variables. The logical record is written into one sector or a number of contiguously numbered sectors, but never less than one sector. Since the 2200 system regards one 256-byte sector as the smallest information unit for storage and retrieval, all data transfers to and from disk are done in 256-byte (one-sector) blocks.

With the DATASAVE DC and DATASAVE DA statements, system control bytes are inserted as the record is buffered to be written to disk; with the DATALOAD DC and DATALOAD DA statements, they are removed as the values just read from disk are equated to the user's argument list. Control bytes are thereby detectable only on disk and include two bytes at the beginning of each sector, one SOV (start-of-value) byte preceding each field (variable or array element), and an EOB (end-of-block) byte following the last field in any sector containing less than 256 bytes of information, as shown in Figure 5-3. The first byte of each sector is byte 000, so the first data byte begins at byte 003. No system control bytes are present with the DATASAVE BA or DATALOAD BA statements whose records contain data beginning at byte 000.

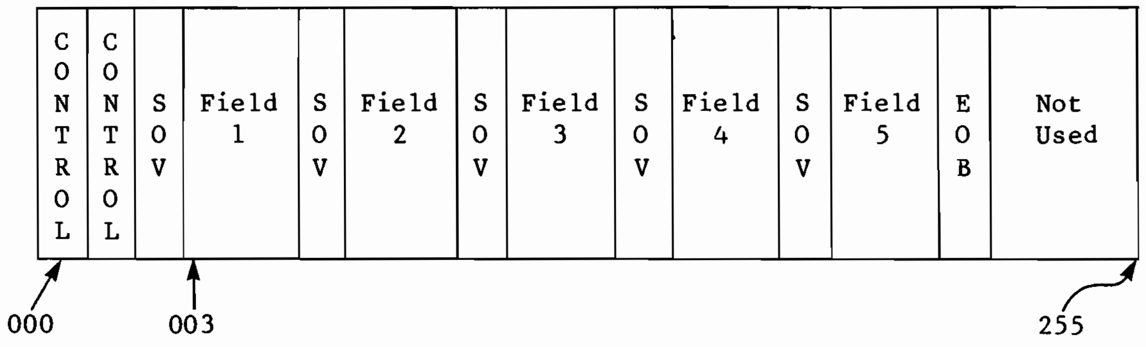


Figure 5-3. Data Sector with Control Bytes

## Logical Record Length and Blocked Records

If the combined argument list field lengths and all control bytes exceed 256 bytes, multiple sectors are read or written using a single DATALOAD (DC or DA) or DATASAVE (DC or DA) statement. With DATASAVE BA and DATALOAD BA statements, however, only one disk sector can be written or read. If the combined argument list field lengths and all control bytes are equal to or less than 256 bytes, only a single sector is required. KFAM supports the use of multiple logical records per sector, one logical record per sector, and also multiple sector logical records.

Since the 2200 system reads or writes a minimum of one sector, special considerations are necessary to handle multiple logical records per sector, or "blocked records." Whether or not blocked records are possible depends upon the logical record length, as calculated according to KFAM conventions.

With DATASAVE BA records which have no control bytes, the logical record length is the sum of all field lengths, based upon the dimensioned length of each variable.* Records may be written as blocked records when the logical record length is less than 128 bytes. To obtain the number of records per sector (an integer), divide the logical record length into 256 and truncate the remainder. Should the calculated logical record length exceed 256, a multiple sector record is required, as described in the Section 5.5.

With DATASAVE DC or DA records, the logical record length is calculated as follows:

1. Add the field lengths, based upon the dimensioned length of each variable.*

2. Add one for each field to account for the SOV control byte which precedes each field; the two sector control bytes (which begin each sector) are not included in this calculation.

3. The sum of the field lengths and SOV control bytes determines the logical record length.

4. Should the calculated record length exceed 254, a multiple sector record will be written, as further described in Section 5.5.

To obtain the number of records per sector, divide the calculated logical record length into 254. The resultant integer (truncate the remainder) is the number of records per sector for the given record length. For instance, if the logical record length is greater than 127 bytes, only one unblocked record can reside in each sector. Two records can reside in each sector for record lengths from 85 through 127 bytes, three records can reside in a sector for record lengths from 64 through 84 bytes, four records can reside in a sector for record lengths from 51 through 63 bytes, and so on.

* Undimensioned scalar numeric-variables are each eight bytes in length; undimensioned scalar alpha-variables are each 16 bytes in length.

The programmer can reduce the logical record length by "packing" some or all of the record's fields into a larger alpha-variable of up to 124 bytes, by using the $PACK statement (or the MATCOPY statement, STR function, or the & concatenation operator). Packing fields reduces the number of SOV control bytes and the logical record length. Packing a record is especially desireable when it enables a multiple sector record to become a single sector record or increases the number of records per block, since this conserves disk space. If the records will be sorted using the SORT-4 Disk Sort Subsystem, it is recommended that the field form of the $PACK statement be used if the $PACK statement is employed (see Chapter 4, Section 4.4 for additional information on SORT-4). A record is typically packed (using $PACK) prior to a DATASAVE statement and unpacked (using the $UNPACK statement) following a DATALOAD statement. (See the BASIC-2 Language Reference Manual for an explanation of the $PACK and $UNPACK statements, the STR function, the MATCOPY statement, and the & operator.)

## 5.5 KFAM RECORD TYPES

KFAM supports five User File record types. Two factors that usually determine the KFAM record type are: (1) whether the records are written with control bytes using DATASAVE DC or DATASAVE DA statements, or without control bytes using a DATASAVE BA statement, and (2) whether the records are blocked records, single sector records, or multiple sector records. Table 5-2 lists the five record types according to the logical record length (as described in Section 5.4) and the DATASAVE mode, which is either DC, DA, or BA.

Table 5-2. KFAM Record Types

| LOGICAL RECORD LENGTH | DATASAVE MODE | KFAM RECORD TYPE |
|---|---|---|
| 127 or less (blocked) | DC or DA | A or C* |
| 128 or less (blocked) | BA | B (C with restrictions) |
| from 128 to 254 | DC or DA | N |
| from 129 to 256 | BA | N |
| greater than 254 | DC or DA | M (multiple sector) |
| greater than 256 | BA | M* (multiple sector) |

*Record type M, BA mode, is not be acceptable to the SORT-4 Disk Sort Subsystem; record type C may not be acceptable to SORT-4.

In each of the six possible combinations, there is usually only one possible record type, with one major exception. With DC or DA mode blocked records, there are two possible record types (A or C) where the records are written either in array or non-array form as specified in the DATASAVE statement argument list. The differences and the reasons for choosing one or the other are discussed below.

Each of the five record types are described in the context of certain parameters required by the Initialize KFAM File utility. These parameters include the starting position of the key, the key length, the number of records per sector, the number of sectors per record, and certain restrictions on the key, which must be from two through 30 alphanumeric bytes stored on disk. (Numeric key field data may be packed using the $PACK statement into alphanuimeric form before being written to disk.)

## User File Record Type A, Array Blocked Records

Record type A, array blocked records, can be written using DATASAVE DC or DATASAVE DA statements where all variables in the argument list are arrays with an equal number of row elements. The row element subscript in each array indicates the record number within the block. That is, Record Number 1 is contained within Array Element 1 of all argument list arrays, Array Element 3 of all arrays contains Record Number 3, etc. The use of arrays allows a numeric scalar variable to be substituted for the subscript number to facilitate efficient processing by setting the numeric scalar variable equal to the appropriate record number, for instance, within a FOR...NEXT loop.

When dimensioning the arrays to be used as arguments, the number of records per sector (block), as determined by the logical record length, is usually equal to the number of array elements. For instance, to write four records where the logical record length is 59, the following statements may be used:

```
20 REM Dimension DATASAVE arguments
30     DIM A$(4)12, B(4), C$(4)36
    .
    .
80 REM Save the Sector Using DATASAVE
90     DATASAVE DC #2, A$(), B(), C$()
```

The block of four records is written noncontiguously within a sector; that is, all elements of array A$() are sequentially written, followed by all elements of array B(), followed by all elements of array C$(). Following the two sector control bytes, each array element is written preceded by an SOV control byte.

Initialize KFAM File requires the following data-related parameters for this record type: (1) the logical record length (see Section 5.4), (2) the blocking factor; i.e., the number of records per sector, (3) the key length, and (4) the starting position of the key. The key length and starting position of the key must be specified so that the key is entirely contained in one alphanumeric field and in the same position of each record (see below). The key may be part of a field (i.e., STR(C$,11,10), but it cannot be a numeric field nor can it contain control bytes. These restrictions can easily be avoided simply by packing the field(s) containing the key, or the entire record, into an alpha-array. The key must be within the same position of each record, and all records must be the same length.

The starting position of the key is calculated to include the two control bytes which begin each sector and all SOV control bytes, as if each of the records were stored contiguously as the only record in the sector. With the above example, the sector written is shown in Table 5-3.

Table 5-3. Sector Layout for Record Type A Example

| BYTES | CONTENTS | BYTES | CONTENTS |
|---|---|---|---|
| 0,1 | Sector control bytes | 73-80 | B(3) |
| 2 | SOV control byte | 81 | SOV control byte |
| 3-14 | A$(1) | 82-89 | B(4) |
| 15 | SOV control byte | 90 | SOV control byte |
| 16-27 | A$(2) | 91-126 | C$(1) |
| 28 | SOV control byte | 127 | SOV control byte |
| 29-40 | A$(3) | 128-163 | C$(2) |
| 41 | SOV control byte | 164 | SOV control byte |
| 42-53 | A$(4) | 165-200 | C$(3) |
| 54 | SOV control byte | 201 | SOV control byte |
| 55-62 | B(1) | 202-237 | C$(4) |
| 63 | SOV control byte | 238 | EOB control byte |
| 64-71 | B(2) | 239-255 | Unused |
| 72 | SOV control byte | | |

The fact that there are four records per sector should be ignored when calculating the starting position of the key. The sector should be viewed as if it contained only one record:

| BYTES | CONTENTS |
|---|---|
| 0,1 | Sector control bytes |
| 2 | SOV control byte |
| 3-14 | A$(1) |
| 15 | SOV control byte |
| 16-23 | B(1) |
| 24 | SOV control byte |
| 25-60 | C$(1) |
| 61 | EOB control byte |
| 62-255 | Unused |

If records are stored as a single array, the starting byte relative to 000 for each blocked record can be derived from the following formulae where: P=starting byte within the block, L=logical record length, and Q=record number within the block:

As stored in the array: P=(Q-1)*(L-1)

As stored in a disk sector: P=(Q-1)*(L)+3

Record type A is usually preferred over record type C since it is always acceptable to SORT-4 and is often more efficient. The major reason for type C records is that they can have keys which span fields and contain control bytes, whereas type A records cannot have keys which span fields or contain control bytes, since each record is noncontiguous; however, these restrictions are easily avoided by packing the fields containing the key with type A records.

## User File Record Type C, Contiguous Blocked Records

Record type C, contiguous blocked records, can be written using DATASAVE DC or DATASAVE DA statements where all variables in the argument list are written sequentially and are typically scalar variables. The DATASAVE BA statement may be used, but the file cannot be reorganized using the Reorganize In Place utility; therefore, type B records are usually specified instead. The records must be the same length, and the key must be located within the same position of each record.

All fields are stored contiguously on disk. For instance, the following statements may be used to write three records where the logical record length is 78:

```
20 REM Dimension DATASAVE Arguments
30     DIM K1$12, D1$64, K2$12, D2$64, K3$12, D3$64
   .
   .
80 REM Save the Sector Using DATASAVE
90     DATASAVE DC #4, K1$, D1$, K2$, D2$, K3$, D3$
```

The block of three records are written within a sector contiguously, and the fields of each record are also contiguous since each argument is written in sequential order.

Initialize KFAM File requires the following data-related parameters for this record type: (1) the logical record length (see Section 5.4), (2) the blocking factor; i.e., the number of records per sector, (3) the key length, and (4) the starting position of the key. The key length and starting position of the key must be specified such that the key is not a numeric field, but it may be part of a field, span fields, and include control bytes.

The starting position of the key is calculated as if there were only one record in the sector. With DC or DA mode records, the first disk data byte always begins at byte 003, the second field in the record begins at the value of: 3 + length of the first field + 1, etc. (see Figure 5-3). With BA mode records, the first data byte always begins at byte 000, and the second field begins at the value of the length of the first field, etc.

## User File Record Type B, BA Mode Blocked Records

Record Type B, BA mode blocked records, is similar to Type C records, but can be written only in BA mode. Both are contiguous blocked records, where all records must have the same length, and the key must be located in the same position of each record. The DATASAVE BA statement is required to write the record, and the DATASAVE BA argument list should resemble that of type C records. Arrays may be used only if each record is entirely contained within a single array element.

The data-related parameters required for Initialize KFAM File are the same as discussed above for Type C records written in BA mode. However, keep in mind that control bytes are not present.

The starting byte location of each record relative to 000 stored in a disk sector or in a single array is as follows, where P=starting byte within the block, L=logical record length, and Q=record number within the block:

P=(Q-1)*L

## User File Record Type N, Single Sector Records, Not Blocked

Record Type N, single sector records which are not blocked, consists of contiguous fields located in one sector, written using either the DATASAVE DC or DA statements (with control bytes) or the DATASAVE BA statement (without control bytes). If arrays are used, elements of each array should be viewed as contiguous, since this is how they are written on disk. The key, which is defined during Initialize KFAM File by the starting position of the key and key length, must be located in the same position of each record and must not be numeric, but it may be part of a field, span fields, and include control bytes.

Each record must occupy one sector or less. For instance, consider the following record whose logical record length is 169 bytes:

```
20 REM Dimension DATASAVE argument list
30     DIM A$25, B, B$40, C, C$40, D$40
   .
   .
80 REM Save record using DATASAVE
90     DATASAVE DC #6, A$, B, B$, C, C$ ,D$
```

Initialize KFAM File requires the following data-related parameters: (1) logical record length (see Section 5.4), (2) the key length, and (3) the starting position of the key.

The starting position of the key is calculated the same as with type C records. With DC or DA mode records, the first disk data byte always begins at byte 003, the second field in the record begins at the value of: 3 + length of the first field + 1, etc. (see Figure 5-3). With BA mode records, the first data byte always begins at byte 000, and the second field begins at the value of the length of the first field, etc.

## User File Record Type M, Multiple Sector Records

Record type M, multiple sector records, can be written using either DATASAVE DC or DA statements (with control bytes) or DATASAVE BA statements (without control bytes). Type M, BA mode records are not acceptable to the SORT-4 Subsystem. If arrays are used, elements of each array should be viewed as contiguous, since this is how they are written on disk. The key, which is defined during Initialize KFAM File by the starting position of the key and key length, must be located in the same position of each record, must not span sectors, and must not be numeric. It may be located in any sector, be part of a field, span fields, and include control bytes.

Records may be up to 255 sectors in length. However, to run the Reorganize In Place utility, records must not exceed 40 sectors in length and, if records exceed eight sectors, the memory required for reorganization will exceed the memory required for KFAM-7 utilities.

With multiple sector DC or DA mode records, whose logical record length exceeds 254 bytes, the 2200 system writes only whole fields in a sector. Any field that cannot be entirely written at the end of a sector is automatically written at the beginning of the next sector. For this reason, argument list variables should be arranged to maximize the amount of data stored in each sector.

Unlike the DATASAVE DC or DA statements where multiple sectors can be written using one DATASAVE statement, one DATASAVE BA statement is necessary for each sector to be written. Additionally, for each DATASAVE BA statement required to write one record, the DATASAVE argument list variables must be carefully arranged such that each field is entirely written within one sector to avoid truncation.

For example, consider the following two sector record as defined by the following statements:

```
20  REM Dimension DATASAVE arguments
30      DIM D$(6)64
     .
     .
80  REM Save the record using DATASAVE
90      DATASAVE DC #8, D$()
```

The layout of this two sector record appears in Table 5-4.

Table 5-4. Record Layout for Record Type M Example

| FIRST SECTOR | | SECOND SECTOR | |
|---|---|---|---|
| Bytes | Contents | BYTES | CONTENTS |
| 0,1 | Sector control bytes | 0,1 | Sector control bytes |
| 2 | SOV control bytes | 2 | SOV control bytes |
| 3-66 | D$(1) | 3-66 | D$(4) |
| 67 | SOV control byte | 67 | SOV control byte |
| 68-131 | D$(2) | 68-131 | D$(5) |
| 132 | SOV control bytes | 132 | SOV control bytes |
| 133-196 | D$(3) | 133-196 | D$(6) |
| 197 | EOB control byte | 197 | EOB control byte |
| 198-255 | Unused, insufficient space for D$(4). | 198-255 | Unused |

Data-related parameters for Initialize KFAM File are: (1) the number of sectors per record, (2) the key length, and (3) the starting position of the key. The number of sectors per record, and not the logical record length, is required. If the key is located within the first sector, the starting position of the key is identical to type C records. However, if the key is located in a second or subsequent sector, add 256 for each sector preceding the sector containing the key and add the starting position of the key within the sector. In the above example, if the key resides in the first eight bytes of D$(4), the starting position of the key is 259 (not 198, which might be calculated by ignoring the way the 2200 system writes multiple sector records).

## 5.6 GENERAL REQUIREMENTS AND CONVENTIONS

General requirements for KFAM files include User File requirements, Key File requirements, file name conventions, certain conventions about the key, and requirements for copying KFAM files. Specific requirements related to programming procedures are described in Chapter 7.

## User File Requirements

If an existing file is to be initialized as a KFAM-7 User File, it must be a cataloged data file. Existing files which contain data records must contain an END (end-of-data) control sector if the key is packed or if the last key is not known when the Build Key File utility is run. The records need not be in ascending order of their keys. If cataloged, the User File must contain a sufficient amount of sectors for all data records and the three control sectors (END, dummy END, and catalog trailer). A cataloged data file may or may not contain data records, and the Initialize KFAM File utility can catalog an uncataloged User File if sufficient disk space exists.

In all cases, the User File must be contained entirely on one disk surface. (See Chapter 8, Section 8.4 for techniques necessary to split one User File into two User Files.) All records within a User File must adhere to the requirements of the same record type, as described in Section 5.5. Variable length records are not supported by KFAM.

All deleted records and, with blocked records, inactive records in a partially full sector should be flagged as inactive with a value of hexadecimal FF in the first byte of the key. Also, all files must be closed at the conclusion of processing, which writes certain recovery information into the User File. Both of these conventions are necessary to run the Key File Recovery utility, which recreates a Key File based upon this information.

## Key File Requirements

If cataloged, a Key File, must contain sufficient disk space. If uncataloged, Initialize KFAM File can catalog the Key File if sufficient disk space exists. In either case, the number of sectors needed is calculated as follows:

1. Calculate the maximum number of KIEs per KIR, using the following formula:

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

   where: N = the maximum number of KIEs per KIR,
   K = the key length, and
   3 = the pointer length (sector address).

2. Calculate the average number of KIEs per KIR (A), based on the value of N from the first calculation:

   A = INT(N*.6)

3. Calculate the number of sectors (S) required for the Key File using the value of A from the previous calculation and the estimated number of records to be contained in the User File (R):

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

Multiple Key Files may be maintained in conjunction with User File records to provide alternate logical key paths and alternate key retrieval capabilities. In the case of real-time applications using multiple Key Files, sufficient space must exist for multiple Key Files, since all Key Files must be on-line whenever User File records are to be added or deleted unless special programming techniques are employed.

## File Name Conventions

The name of any User File is always in the format "bbbbFnbb", where any b can be replaced by any character, n must be a digit (from 0 through 9), and an F must be in the fifth position. The name of any Key File has the identical sequence of characters as its corresponding User File, but a K in the fifth position replaces the F, and the digit in the sixth position is the Key File number, which is required during during Initialize KFAM File. Any cataloged User File and Key File must conform to these file name characteristics, or it cannot be initialized by Initialize KFAM File. For example, the User File DATAF100, which has a Key File whose number of 1 must have a Key File name of DATAK100.

## The Key

The presence of a key within each User File record allows KFAM to index each record within a Key File. The key value of a record must be known in order to delete a record, to locate a given record based on its key value (instead of searching the entire file), and to add a new record. Especially for those applications involving record retrieval and updating, the chosen key should be easy to remember or available on a list for the operator (or application program) to reference. Each key must have a unique value for each User File record or special programming considerations must be implemented.

Each key may be from two to 30 bytes of alphanumeric data, including hexadecimal data and packed numbers. Numeric keys are not allowed. Requirements for the key's location vary with the record type used (see Section 5.4). Duplicate keys are not allowed. Key sequence is based upon the 2200 ASCII collating sequence (character code assignments).

The first byte in an active record's key must not contain the value of hexadecimal FF, since that value is reserved to indicate (flag) deleted or inactive User File records. Similarly, the key may not have a value of all bytes hexadecimal 00 (zero), which is reserved for KFAM use.

Where multiple Key Files are employed to index a single User File, the record's key indexed by each Key File is typically located within different positions of a User File record. One of the Key Files is a "Primary Key File" and its key is referred to as the primary key, which must be unique for all records within the User File. Other Key Files are referred to as "Secondary Key Files," and their keys are referred to as secondary keys. Secondary (alternate) keys need not be unique if certain procedures are employed, which are described later in this manual.

## Closing KFAM Files

To retain accurate control information, recovery information, and to avoid unnecessary delays in file access, each KFAM User File/Key File should be closed at the conclusion of file access. In addition to the automatic closing of files upon completion of application program execution, an operator is typically given an SF key which can be touched during program execution to close the files, allowing unscheduled termination of the application program.

## Copying KFAM Files

Backup copies should be made of each KFAM User File and Key File regularly. To copy an entire disk surface, the COPY or MOVE statement may be used, followed by the VERIFY statement (see the Wang BASIC-2 Disk Reference Manual). Alternatively, the ISS Copy/Verify utility may be used to copy all or specified files on a file-by-file basis from one disk surface to another. (a form of the MOVE statement also allows specified files to be copied from one disk surface to another.)

KFAM maintains an END (end-of-data) control sector in the User File and the Key File. KFAM maintains critical system information within the Key File which is directly related to the position of both END sectors, as well as recovery information in the next-to-last sector in the User File. To ensure that this system information is not lost or changed when using the Copy/Verify utility, the specified extra sector value should be the same as the number of extra sectors in the input User File and Key File. The Copy/Verify utility allows an extra sector value of -1 to be specified, which indicates that the same number of extra sectors in the input file are to be copied into the output file .

To lengthen or shorten the size of a User File or Key File, either the Copy/Verify utility or the Reorganize/Rebuild Subsystem utility should be used to increase or decrease the number of extra sectors in the User File or Key File (or both). After completing copy using the Copy/Verify utility, the Reallocate File Space utility must be run on the output User File and Key File to reset the critical system information in the Key File which maintains the position of END (end-of-data) control sector in the User File or Key File. If the Reorganize/Rebuild Subsystem is used, Reallocate File Space is not needed.

NOTE:

An extra sector value of 0 (zero) should never be specified during copy of a User File, since this value does not allow sufficient space for the recovery information in the next-to-last User File sector to be copied.

## Changing the Record Layout of an Existing KFAM File

It may become necessary to modify the record layout of a KFAM User File, for instance, to add a new field or increase the size of an existing field. Procedures recommended for changing a KFAM file using a simple application program are provided in Chapter 8, Section 8.9. The procedure involves reading the old file and writing the modified records into a new file.

## Use of Duplicate Keys

A special KFAM convention allows the KFAM requirement of unique keys to be simulated by concatenating a unique KFAM-supplied pointer onto each key used as a subroutine argument. This convention allows duplicate keys to be present for User File storage and user-interface purposes, while meeting the KFAM requirement of unique keys in the Key File. The Build Key File utility and the Reorganize/Rebuild Subsystem utility are available with an option which builds the Key File index by using the key-pointer concatenation convention. User-supplied programs may add records using similar special programming techniques, which are also required for random record access (by key) operations. This advanced feature is described in Chapter 7, Section 7.4.

## Use of Multiple Key Files

The use of up to nine Key Files is supported by Wang Laboratories (although only eight can be open simultaneously) if the requirements and conventions described in Chapter 6, Section 6.1 (for utility use) and Chapter 7, Section 7.4 (for subroutine use) are observed. Use of multiple Key Files allows multiple keys to be used to index User File records. As an alternative to sorting operations for such tasks as report generation, each key can be used to output records according to ascending or descending key sequence of the chosen key, or records may be retrieved randomly based upon any one of the keys. The efficient searching capabilities provided by multiple keys include alphanumeric and numeric range search and other custom-tailored applications under user-supplied program control. Duplicate key use with multiple keys is supported and thereby expands the scope of potential applications.

Multiple Key File use requires certain special programming procedures recommended only for advanced programmers with previous experience writing KFAM-7 application programs. Note that a user-designed record protection scheme must be implemented if the Multiplexed KFAM-7 version is to be used. (The different KFAM-7 versions are defined and described in Section 5.7).

In order to reorganize multiple Key Files and to recover from Key File destruction using the Key File Recovery utility, the User File must contain a valid END control sector and valid recovery information typically for the Primary Key File.

## 5.7 CHOOSING THE CORRECT KFAM-7 VERSION

Three versions of KFAM-7 are provided to accommodate various hardware environments and user needs. The storage and maintenance of KFAM file/record access control information differs for each version, but these differences are automatically handled internally by KFAM subroutines and are unseen by the user's software. Programming and KFAM utility operation for the different versions are nearly identical, and the Key File's KIRs and KIEs are always maintained and searched by KFAM in the same manner. Table 5-5 lists and describes the three KFAM-7 versions.

Table 5-5. KFAM-7 Versions and Applicable Hardware

| KFAM-7 VERSION | APPLICABLE HARDWARE |
|---|---|
| Multiplexed Version | This version is typically used with a 2200VP central processor. This version is also required for each 2200VP or 2200MVP central processor connected via a disk multiplexer to the same disk drive containing KFAM-7 files accessible to other CPUs. It may optionally be used on a 2200MVP where only one station will access KFAM-7 files and global subroutine use is not desired. This version is used by KFAM utilities if the global partition "KFAM" is not present in that memory bank. This version does not fully support multiple Key Files; the user must implement a record protection scheme (see Chapter 7, Section 7.4). |
| Single Bank Version | This version is typically used on a 2200MVP central processor with 64K memory or less (Single Bank 2200MVP). It can also be used on a 2200MVP with more than 64K memory (Multiple Bank 2200MVP) if all stations accessing KFAM-7 files are in one and only one memory bank. This version must not be used with a 2200VP or 2200MVP multiplexed to a disk drive containing shared KFAM-7 files. This version fully supports multiple Key Files, including record protection for multiple Key Files. |
| Multiple Bank Version | This version is usually used only on a Multiple Bank 2200MVP (more than 64K memory). This version must not be used with a 2200MVP multiplexed to a disk drive containing shared KFAM-7 files. This version fully supports multiple Key Files, including record protection for multiple Key Files. |

## Multiplexed Version

The Multiplexed version consists of the KFAM subroutines and variables contained in program file KFAM0207 and the KFAM user variables in program file KFAM0007. If global 2200MVP operation is desired, KFAM0207 is loaded and run in an 8K memory partition (including partition overhead) for each memory bank accessing KFAM-7 files. Each user-supplied program accessing KFAM-7 files must have incorporated into it the KFAM user variables in KFAM0007; these variables require from 1K to 2K memory depending upon the number of files to be accessed simultaneously (not including partition overhead). With nonglobal (e.g., 2200VP) operation, both the KFAM0207 and KFAM0007 program files are incorporated into the user-supplied program.

This version stores and maintains file/record access information in the Key File, which is read and written more frequently than with other KFAM-7 versions. The Key File's KDR serves as a central communication link among multiple users; this disk-resident link allows multiple CPU's to access the same KFAM-7 files simultaneously. Record protection for multiple Key Files is not supported, since record access information is maintained in each individual Key File's KDR, and not in one centralized location; e.g., portion of memory.

## Single Bank Version

The Single Bank version consists of KFAM subroutines and variables contained in program file KFAM0107 and the KFAM user variables in program file KFAM0007. Global operation occurs after KFAM0107 is loaded and run in a 9.75K partition (including partition overhead). Each user-supplied program accessing KFAM-7 files must have incorporated into it the KFAM user variables in KFAM0007; these variables require from 1K to 2K memory depending upon the number of files to be accessed simultaneously (not including partition overhead). With nonglobal (e.g., 2200VP) operation, both the KFAM0207 and KFAM0007 program files are incorporated into the user-supplied program, and only that user must be accessing KFAM files.

This version stores and maintains file/record access control information in certain global variables (not found in the Multiplexed version's KFAM0207). Global KFAM variables serve as the central communication link for multiple users, allowing stations within one 2200MVP memory bank to access the same KFAM files simultaneously. Since file/record access control information is dynamically maintained in global memory, the Key File is read and written less frequently than with the Multiplexed version, and multiple Key File record protection is fully supported. Consequently, the Single Bank version provides better performance than the Multiplexed version. A global queue ensures that multiuser subroutine execution occurs on a first-in-first-out basis. With nonglobal operation, this version allows a single station to access KFAM files without the disk access to the KDR associated with the Multiplexed version.

## Multiple Bank Version

The Multiple Bank version consists of the KFAM subroutines contained in program file KFAM0307, the KFAM variables in program file KFAM0407, and the KFAM user variables in program file KFAM0007. Although this version can operate on a Single Bank 2200MVP, it is designed for Multiple Bank 2200MVP use. The KFAM variables in KFAM0407 are loaded and run in the universal global memory area and require a 2.75K partition (including partition overhead). The KFAM subroutines in KFAM0307 are loaded and run in an 8.5K partition (including partition overhead) within each memory bank in which stations may access KFAM-7 files. Each user-supplied program accessing KFAM-7 files must have incorporated into it the KFAM user variables in KFAM0007; these variables require from 1K to 2K memory depending upon the number of files to be accessed simultaneously (not including partition overhead).

This version stores and maintains file/record access control information in certain global KFAM variables (not found in the Multiplexed version's KFAM0207). Like the Single Bank version, global KFAM variables serve as the central communication link for multiple users. However, since the global variables are stored separately in the universal global area, they allow stations within all 2200MVP memory banks to access the same KFAM files simultaneously. Since file/record access control information is dynamically maintained in universal global memory, the Key File is read and written less frequently than with the Multiplexed version, and multiple Key File record protection is fully supported. The Multiple Bank version provides comparatively better performance than the Multiplexed version. A global queue ensures that multiuser subroutine execution occurs on a first-in-first-out basis.

## Changing the Maximum Number of KFAM Files to be Open Simultaneously

The KFAM-7 variables contained in the global modules KFAM0107, KFAM0207, or KFAM0407 include two global arrays which handle certain record access functions, per KFAM file. Should the user need to have more or less than 30 KFAM files open simultaneously, the array elements (preset to 30) in the COM statement for arrays @T$() and @V4$() may be changed to the desired value; however, the required global partition size will increase or decrease, depending on the array element number specified in the COM statement for @T$() and @V4$().

The modified program file can be saved into a separate program file, for instance, on the 2200MVP operating system diskette, so it can be automatically loaded and run upon partition execution by the automatic bootstrap feature.

## 5.8 GETTING 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 and moves that station's Current Sector Address to the sector containing that record, the processing of the record (loading it, updating it, saving it, etc.) is left to the user-written program. Thus, one must have a working knowledge of elementary BASIC-2 and the fundamentals of Catalog Mode disk operations to use KFAM.

It is strongly recommended that first-time KFAM users begin by setting up a dummy KFAM 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 each KFAM file.

1. Read Sections 5.5 and 5.6 which describe the five types of User File records acceptable to KFAM, limitations on the size and characteristics of the key field, and certain KFAM conventions.

2. A Key File is stored as a cataloged file on a disk. It may reside either on the same disk as the User File or on another disk, which must be mounted whenever the User File is accessed. The Initialize KFAM File utility must be run to save certain information in the Key File, based upon information supplied by the operator. However, if a KFAM-3 or -4 file is to be converted to KFAM-7 format, Initialize KFAM File is not usually required; use the Convert to KFAM-7 utility instead.

3. If a previously cataloged User File contains data, a second setup utility called Build Key File is run following Initialize KFAM File to build the Key File index based upon the User File records. After running Build Key File, KFAM subroutines may be used to access User File records and perform file maintenance tasks. For files converted to KFAM-7 format using the Convert to KFAM-7 utility, Build Key File is not usually necessary, and KFAM-7 subroutines may be used to access the converted 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 add the corresponding entries into the Key File; KFAM subroutines can thereafter be used to perform file maintenance tasks.

5. Once records reside in the KFAM file, it is recommended that the Print Key File utility be run to ensure that Key File control information was correctly defined during Initialize KFAM File.

6. The KFAM subroutines are DEFFN' statement subroutines which perform standard tasks for files indexed by KFAM. The three versions of KFAM-7 have different subroutine program files, as listed in Section 5.7. The subroutines contained in KFAM0207 (Multiplexed), KFAM0107 (Single Bank), and KFAM0307 (Multiple Bank) include all KFAM-7 subroutines. Alternatively, the user may select the subroutines needed for the application(s) to be performed using the Build Subroutine Module utility, which creates a module (program file) containing the chosen subroutines corresponding to the specified KFAM-7 version. The module created by Build Subroutine Module may be used instead of the supplied module if certain considerations are observed. Build Subroutine Module is helpful in reducing memory requirements, especially for nonglobal (e.g., 2200VP) programs where only those subroutines needed by that particular user-supplied program need to be included. (See Chapter 6, Section 6.9 for additional information.)

7. Where global 2200MVP operation is desired, certain considerations related to partition generation are necessary. As described in the 2200MVP Introductory Manual, partition generation is the allocation of system resources, including memory, to various terminals. Before KFAM-7 utilities and user-supplied programs can access the KFAM-7 subroutines and variables, the appropriate global partition(s) must be loaded and run. With the Multiplexed and Single Bank versions, one global "KFAM" partition is needed per bank. With the Multiple Bank version, one global (subroutine) KFAM partition is required in each bank and one global (KFAM variables) KFAMCOM partition is required in universal global memory. Refer to Section 5.7 for required partition sizes, file names, and related information.

8. The KFAM-7 subroutines are listed and briefly described in Table 5-1. Chapter 7 of this manual describes each subroutine in detail and provides programming guidelines, procedures, and techniques associated with the subroutines.

9. Although the KFAM subroutines are the heart of the KFAM system and perform most of the file maintenance, utilities are included to perform certain tasks that will occasionally be required. The KFAM-7 utilities are listed and briefly described in Section 5.3. Chapter 6 of this manual describes each KFAM-7 utility and provides certain operating guidelines.

# CHAPTER 6
# THE KFAM-7 UTILITIES

## 6.1 INTRODUCTION

The KFAM-7 utilities provide a variety of functions related to KFAM-7 files, including the following:

- A User File and Key File can be set up based on operator-entered information. The Key File index can be built if the User File previously contained data records. These functions are provided by the setup utilities, Initialize KFAM File and Build Key File. (Reorganize/Rebuild Subsystem may be used instead of Build Key File.)

- A KFAM-3 or -4 file may be converted to KFAM-7 file format using the conversion utility, Convert to KFAM-7.

- The contents of the Key File may be printed or displayed to verify that the file was set up using the correct information. The Key File information provided by the Print Key File utility is useful as a diagnostic tool, especially when delays in accessing KFAM-7 files become excessive due to access mode conflicts.

- KFAM-7 User Files and Key Files may be increased or decreased in size during copy if the Reallocate File Space utility is run following the copy operation.

- When a record is deleted, its key and sector address are removed from the Key File, which makes the User File record inaccessible; however, the data record itself is not removed from the User File. The supplemental maintenance reorganize utilities, Reorganize In Place and Reorganize/Rebuild Subsystem, remove deleted records and place each record in ascending key sequence. Reorganize/Rebuild Subsystem can optionally rebuild a Key File, similar to the function provided by Build Key File.

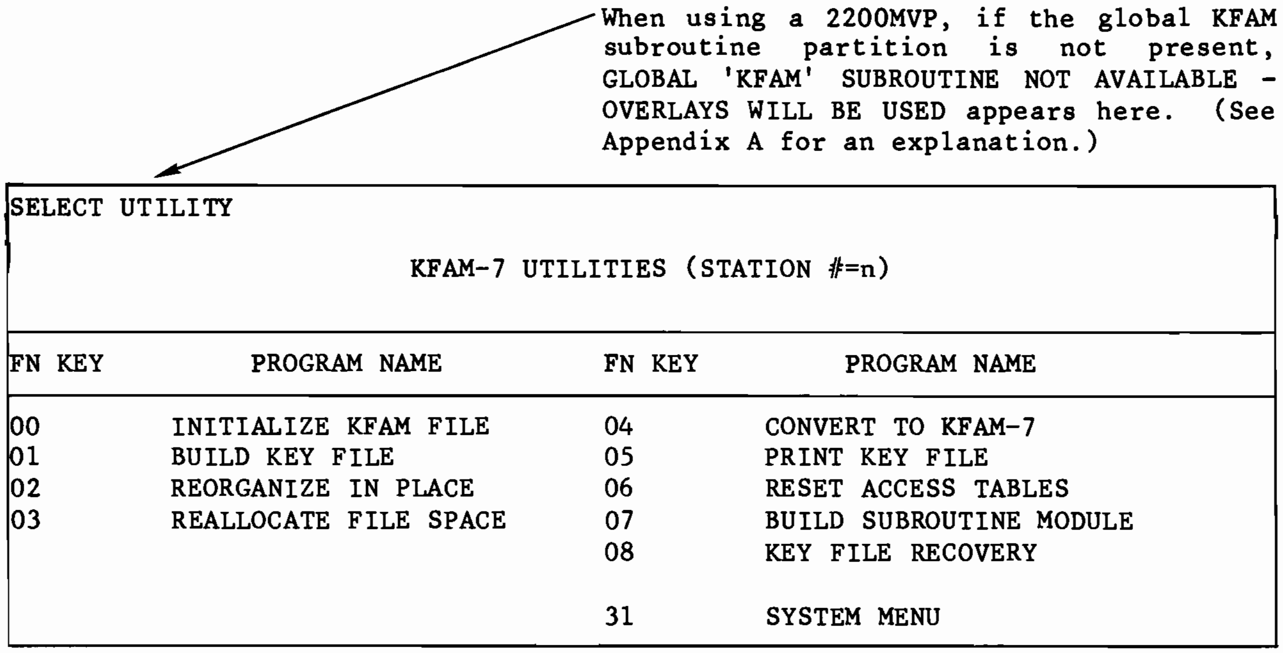
- A subset of subroutines may be chosen, saved to disk in a program file, and used instead of the complete set of subroutines provided for each KFAM-7 version, by using the Build Subroutine Module utility.

- In the event of a Key File being accidentally destroyed or KFAM files being accidentally left open (which renders them inaccessible due to access mode conflicts), the recovery utilities, Key File Recovery and Reset Access Tables, are provided.

## The KFAM-7 Menu

The KFAM-7 menu, as shown in Figure 6-1, may be obtained following start-up by either specifying KFAM-7 as the MENU TO LOAD or touching the appropriate SF key in reply to the System menu. Memory requirements (2200MVP partition sizes) for the KFAM-7 Utilities are provided in Table 1-1.

When using a 2200MVP, if the global KFAM subroutine partition is not present, GLOBAL 'KFAM' SUBROUTINE NOT AVAILABLE - OVERLAYS WILL BE USED appears here. (See Appendix A for an explanation.)



```
SELECT UTILITY

                       KFAM-7 UTILITIES (STATION #=n)

FN KEY          PROGRAM NAME           FN KEY          PROGRAM NAME

00        INITIALIZE KFAM FILE         04        CONVERT TO KFAM-7
01        BUILD KEY FILE               05        PRINT KEY FILE
02        REORGANIZE IN PLACE          06        RESET ACCESS TABLES
03        REALLOCATE FILE SPACE        07        BUILD SUBROUTINE MODULE
                                       08        KEY FILE RECOVERY

                                       31        SYSTEM MENU
```

Figure 6-1. The KFAM-7 Utilities Menu

To load one of the KFAM-7 utilities, touch the appropriate SF key. To load the System menu, touch SF'31. The Reorganize/Rebuild Subsystem utility does not appear on the KFAM-7 menu, since it is controlled by a user-supplied program.

## KFAM-7 Utility Default Values and Operating Similarities

For each utility on the KFAM-7 menu, default values are maintained for each station in its respective station file. When a KFAM-7 utility is loaded, its default values are loaded from the appropriate station file and displayed along with the ENTER DESIRED FUNCTION prompt. Like the ISS start-up default values, each default value appears to the right of a number which allows modification of the default value when its number entered in reply to the ENTER DESIRED FUNCTION prompt. If there are no default values for that utility, entry of each value is requested, beginning with value 1.

After modifying and/or entering the required values and verifying that the displayed values are correct, the user may save the currently displayed defaults (SAVE DEFAULTS) by touching the corresponding number before entering 0 (zero) to proceed. Other prompts may appear after 0 is entered, and the user may touch SF'15 to return the ENTER DESIRED FUNCTION prompt to the screen and modify the default values before utility processing begins (usually preceded by a MOUNT DISK prompt).

If the utility default values for one or more disk addresses conflict with the current ISS start-up disk addresses, entry is automatically requested when that utility is loaded. Whenever a disk address is requested, valid start-up disk addresses are displayed.

## Error Messages and the Start-up Printer Address

During the entry of parameters in reply to prompts, any error message encountered appears on the screen. Once utility processing (execution) has begun, however, error messages are output to the printer address specified during start-up for that station. Refer to Chapter 1, Section 1.4 for additional information.

## Special KFAM Utility Considerations for Multiple Key Files and Duplicate Keys

Readers not using multiple Key Files or duplicate keys should skip to Section 6.2.

One Key File is typically designated as a Primary Key File whose associated primary key might be a social security number, employee number, part number, or some other unique identification. The User File records, when reorganized, are arranged in ascending order of their primary key values. The primary key is typically unique and, if unique, does not require the pointer-key concatenation considerations required for duplicate keys.

The secondary key associated with each Secondary Key File is usually a different field or portion of the record than other secondary keys and the primary key. A secondary key might be a name, department number, numeric quantity, or description needed to access the records in either random order or logical secondary key sequence. Secondary keys may or may not be unique. For each secondary key which may be duplicate, a three-byte pointer can be appended onto the secondary key in the Key File. The pointer is obtained from KFAM variable T4$, which contains a relative sector address and record number within that sector of the corresponding User File record. Since the pointer is unique, when it is appended to a secondary key, the secondary key stored in the Key File is assured of being unique.

The Initialize KFAM File utility assigns user-specified values to certain parameters stored in the Key File, including User File record layout, key location, and key length. Other user-specified parameters include the Key File number (which becomes part of the Key File name) and such operational parameters as disk addresses. Initialize KFAM File is always the first step in setting up a KFAM file, regardless of whether a Primary or Secondary Key File is to be initialized.

A recommended convention is to assign the Primary Key File a Key File number of 1; for Secondary Key Files, assign Key File numbers of 2, 3, 4, 5, etc. Another special consideration related to Initialize KFAM File concerns Secondary Key Files, where the secondary key may be duplicate. The key length and record layout parameters must account for the presence of the 3-byte pointer to be concatenated onto the secondary key, which can be added to the Key File as records are written via either Build Key File or user-supplied application software.

The Build Key File utility can create the "index" for both primary and secondary Key Files if the User File already contains data records. The availability of the Initialize KFAM File and Build Key File utilities allow the creation of secondary Key Files based upon User File records. Build Key File automatically concatenates the appropriate pointer onto each secondary key in the Key File, if the multiple Key File/duplicate key parameter is specified, allowing duplicate secondary keys to be used.

The user must either update all Secondary Key Files whenever the Primary Key File is updated, or only access Secondary Key Files after running Reorganize/Rebuild Subsystem to reorganize the Primary Key File and update each Secondary Key File to be accessed. If insufficient disk space exists, the Reorganize In Place utility may be run for the Primary Key File, followed by Reorganize/Rebuild Subsystem for each Secondary Key File. In a real-time environment, all Key Files should be updated under program control whenever a record is added or deleted in the Primary Key File. For environments where week-old information or periodic Secondary Key File use is acceptable, a copy of the User File and only the Primary Key File might be on-line constantly, the User File and Primary Key File reorganized and copied to a separate disk, and the Secondary Key Files created using Reorganize/Rebuild Subsystem on the separate disk to allow multiple Key File access to the data with a minimum of special programming considerations.

Table 6-1 lists each KFAM utility and describes the requirements and restrictions associated with multiple Key File use.

Table 6-1. KFAM Utility Use with Multiple Key Files

| KFAM-7 UTILITY | PRIMARY KEY FILE | SECONDARY KEY FILE |
|---|---|---|
| Initialize KFAM File | Required, except for KFAM-3 or -4 files (use the Convert to KFAM-7 utility instead). | Required for each Secondary Key File. |
| Build Key File | Required only if the User File contains data records. | Required for each Secondary Key File only if the User File contains data records. |

Table 6-1. KFAM Utility Use with Multiple Key Files (continued)

| KFAM-7 UTILITY | PRIMARY KEY FILE | SECONDARY KEY FILE |
|---|---|---|
| Print Key File | Recommended after data records have been added to verify that the key's location is correct before initial program testing. | Recommended after data records have been added to verify that the key's location is correct before initial program testing. |
| Reallocate File Space | When the User File or the Primary Key File is copied and its number of extra sectors is changed, this utility is required for the Primary Key File. | When the User File is copied and its number of extra sectors changed this utility is required for all Secondary Key Files. For each Secondary Key File copied where its number of extra sectors is changed, this utility is required. |
| Reorganize In Place | Can be used only for the Primary Key File. Except in cases where insufficient disk space exists, use Reorganize/Rebuild Subsystem instead. | Not allowed. Following reorganization of the Primary Key File, the user must run either the Reorganize/Rebuild Subsystem or the Build Key File Utility for each Secondary Key File. |
| Reorganize/Rebuild Subsystem | Must be executed for the Primary Key File first, and then must be executed for each Secondary Key File. | Must be executed for the Primary Key File first, and then must be executed for each Secondary File. If the Reorganize In Place utility was used for the Primary Key File, run either the Reorganize/Rebuild Subsystem or the Build Key File Utility for each Secondary Key File. |

Table 6-1. KFAM Utility Use with Multiple Key Files (continued)

| KFAM-7 UTILITY | PRIMARY KEY FILE | SECONDARY KEY FILE |
|---|---|---|
| Convert to KFAM-7 | Used in place of Initialize KFAM File and Build Key File for KFAM-3 or -4 files only. Valid only for the Primary Key File. | Not allowed. Use Initialize KFAM File and Build Key file for each Secondary Key File. |
| Build Subroutine Module | Not applicable. | Not applicable. |
| Key File Recovery | Can be used only for the Primary Key File, if it was destroyed. With multiple Key Files, this utility must be run first before running Initialize KFAM File. | Not allowed, since recovery information is maintained only for the Primary Key File (which is the Key File closed with a positive KFAM ID number). Instead, use Initialize KFAM File to recatalog each destroyed Secondary Key File, followed by either the Reorganize/Rebuild Subsystem or the Build Key File Utility for each Secondary Key File. |
| Reset Access Tables | With the Multiplexed KFAM-7 version, this utility is required once for each Key File open at the time of the accident. With the other KFAM-7 versions, this utility may be run once if the mode is ALL and the station number is 0 (zero). | Same as for the Primary Key File. |

## 6.2 INITIALIZE KFAM FILE

The Initialize KFAM File utility must be run as the first step in setting up any KFAM file. If the file already contains data records, Initialize KFAM File should be followed by Build Key File.

Initialize KFAM File optionally catalogs an area on disk for the User File and/or the Key File, or it operates with an existing (active or scratched) User File and/or Key File. It sets up the KDR, which is the first sector of the Key File and contains 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 KIR, in the null Key File's second sector. An END (end-of-data) control sector is written following the first two sectors.

The position of the END control sector in the User File depends upon whether the User File was previously cataloged. With a previously uncataloged file just created, the END control sector is written in the first sector. With a previously cataloged file, the position of the END control sector, if any, remains unchanged. In both cases, recovery information (KDR) is optionally stored in the next-to-last sector (dummy END control sector) of the User File, and the access table and password is stored in the User File catalog trailer control sector. The access table is cleared upon being written.

Initialize KFAM File requires that certain information be supplied by the operator, some of which is not obvious and requires some explanation, as provided in Table 6-2.

Table 6-2. Information Required by Initialize KFAM File

| ITEM | EXPLANATION |
|---|---|
| USER FILE NAME | An F is required in the fifth position and a digit in the sixth position (bbbbFnbb form) as described in Chapter 5, Section 5.6. If the User File is an active or scratched cataloged file, its exact file name is required; if not cataloged, a file name unique to the disk specified is the User File Address as required. |
| USER FILE ADDRESS | The three-character (xyy form) disk address indicates the disk where the User File will reside (or currently resides). |
| USER FILE | Either CATALOGED or NOT CATALOGED is specified, depending upon whether or not the User File is a cataloged (active or scratched) file. |

Table 6-2. Information Required by Initialize KFAM File (continued)

| ITEM | EXPLANATION |
|---|---|
| PASSWORD | If a password was previously assigned to an active cataloged file, it must be specified exactly as previously assigned; otherwise, specify blanks. With a scratched cataloged file about to be reactivated or a "not cataloged" User File about to be created, the specified password is assigned to the User File and must be specified whenever access to the User File is desired. By convention, a password of blanks (touch RETURN without entering any characters) indicates that a password is not used. Up to 16 characters may be specified. |
| KEY FILE NUMBER | A digit from 1 to 9 is specified. Typically, 1 is specified unless multiple Key Files are used. If the Key File is an active or scratched cataloged file, the number entered must correspond with that Key File's number; if not cataloged, the number should allow a unique Key File name to be created on the specified disk. The number (n) specified becomes part of the Key File name (bbbbKnbb form) as described in Chapter 5, Section 5.6. |
| KEY FILE ADDRESS | The three-character (xyy form) disk address indicates the disk where the Key File will reside (or currently resides). |
| KEY FILE | Either CATALOGED or NOT CATALOGED is specified, depending upon whether or not the Key File is a cataloged (active or scratched) file. |
| RECOVERY | Either WRITE or NOT WRITE is specified. typically, when initializing a Secondary Key File, recovery information is not written; for Primary Key Files, it is written. |
| RECORD TYPE | The five KFAM record types are described in Chapter 5, Section 5.5. |
| LOGICAL RECORD LENGTH | Required only for record types A, B, and C, the logical record length may be calculated using the procedures provided in Chapter 5, Section 5.4. Specify 1 for record types M and N. |

Table 6-2. Information Required by Initialize KFAM File (continued)

| ITEM | EXPLANATION |
|---|---|
| BLOCKING FACTOR | Required only for record types A, B, and C, the blocking factor or number of records per sector may be calculated using the procedures provided in Chapter 5, Section 5.4. Specify 1 for record types M and N. |
| SECTORS PER RECORD | Required for record type M only, this indicates the number sectors in a multiple sector record, as discussed in Chapter 5, Section 5.5 under "User File Record Type M, Multiple Sector Records." For other record types, specify 1. |
| KEY LENGTH | The key length and starting position of the key determine the key's location. Conventions and requirements associated with the key depend upon the record type, as discussed in Chapter 5, Section 5.5. In the case of duplicate keys, the key length must include the 3-byte pointer concatenated to each key in the Key File. |
| STARTING POSITION OF THE KEY | See KEY LENGTH. |
| NUMBER OF RECORDS | The estimated number of records determines the number of sectors required for the User File and/or Key File. This estimate should include the probable number of deleted User File records when the User File is at its maximum record size. This value is not critical, since the size of the User File and/or Key File may be changed during file copy. The value entered here is used to allocate file space to the User File and/or Key File about to be created. With cataloged files, the value is used to check if enough space exists in these files. |
| KEY TYPE | Either STANDARD or DUPLICATE is specified. If the key being defined allows duplicate keys, the key location plus its key length may exceed record boundaries (since the three-byte pointer has been included in the key length calculation) DUPLICATE adjusts the checking appropriately |

Operating Notes

The values for the USER FILE and KEY FILE parameters alternate (flip-flop) as either CATALOGED or NOT CATALOGED when the appropriate number is touched in reply to the ENTER DESIRED FUNCTION prompt.

If the Key File is an active cataloged file, after the User File name, Key File number, and Key File address have been specified, the user may touch SF'0 to retrieve the following parameters from the Key File's KDR: the record type, logical record length, blocking factor, sectors per record, key length, and starting position of the key. The retrieved parameters are displayed and may be accepted or modified.

After accepting the default parameters, the user may elect to output to the ISS start-up printer address a hardcopy listing of the specified file parameters this option is especially recommended for Secondary Key Files (if used), since recovery information is not maintained for Secondary Key Files. The KFAM-7 menu appears upon completion of initialization. If an error message appears, refer to Appendix A.

## 6.3 BUILD KEY FILE

The Build Key File utility creates a Key File index 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.

Build Key File ignores any records that have HEX(FF) in the first byte of the key, under the assumption that they are deleted or inactive records. It also ignores records which have duplicate keys unless the user specifies that duplicate keys are allowed, but outputs to the ISS start-up printer address 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 User File record in physical sequence. The program can use this to detect the end-of-data. 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 control sector. Furthermore, with blocked records, all inactive records in any sector must contain the value of hexadecimal (HEX) FF in the first byte of the key. However, if the key is packed before being written to disk, the file must contain an END control sector and, with blocked records, all inactive records in any sector must contain the value of HEX(FF) in the first byte of the key.

The KFAM file is opened in the Exclusive mode.

Operating Notes

Default values include the User File name, User File address, Key File number, Key File address, the last key (if required), key type, and the password (if any) assigned to the User File. If the last key is not required, or if the User File does not have a password, a value need not be entered (or enter blanks) for this parameter. The key type is either STANDARD (as in previous versions of KFAM) which indicates unique keys, or DUPLICATE which indicates that duplicate keys are present and that the 3-byte pointer concatenated onto the key is to be written into the Key File. The key type alternates (flip-flops) as either STANDARD or DUPLICATE when the appropriate number is chosen in reply to the ENTER DESIRED FUNCTION prompt.

In this utility, RECOVERY provides the options WRITE/DON'T USE or USE/DON'T WRITE. Typically, for Secondary Key Files, use (USE/DON'T WRITE) the recovery information found in the User File. Primary Key Files should write (WRITE/DON'T USE) the recovery information. Only the FINDNEW SECTOR portion of the recovery information is used.

After accepting the default values, the record locations and keys of the User File records are displayed during processing. The KFAM-7 menu appears upon completion of the utility. If an error message appears, refer to Appendix A.

## 6.4 PRINT KEY FILE

The Print Key File utility prints or displays the control information stored in the specified Key File. The control information output is labeled with applicable variable names and descriptions.

Three types of control information are output:

1. The contents of the KDR, excluding the access tables, are output. Each variable's name, description, and value are output. Refer to Table 8-1 for a list of KDR information.

2. The contents of the KDR access tables and the User File access table are output per station. The list of protected sectors and completion codes are reflected only for the Multiplexed version of KFAM-7, since only the disk-resident control information, and not global control information, is output. The access mode number (ACCESS TYPE), completion code, protected sector, FINDNEW sector, and FINDNEW record are output per station (1-16); sector addresses are hexadecimal relative sector addresses. The access mode number, per station, is helpful in determining the cause of access mode conflicts. For more information, see Section 6.11.

3. The contents of each KIR, each of which occupy one sector, consists of multiple KIEs. Each KIE consists of a key value and hexadecimal relative sector address. Each KIR, and all KIEs contained within each KIR, are output.

The Key File index structure employs multiple levels of indexing, designed to minimize Key File search time. Each KIR is part of a "tree structure," wherein all KIRs on one index level furnish pointers to the next lower-level KIRs, except for the lowest-level KIRs (Index Level 1). Level 1 KIEs index the actual User File record keys and relative sector addresses, whereas each higher-level KIE contains the highest key and relative sector address of lower-level KIRs.

The KDR variable T2$ is the relative sector address of the highest-level KIR, and the KDR variable T0 indicates the number of index levels (1-8). The user might examine the KIR/KIE structure, as KFAM does, by beginning with the highest-level KIR and following the search path from the highest-level KIR to the Level-1 KIR, which points, via a User File relative sector address, to a particular record's key, as further described in Chapter 8, Section 8.8.

## Operating Notes

Default values are maintained for the User File name, address, and password (if any); the Key File number and address; and the output device. When the ENTER DESIRED FUNCTION prompt appears, entering the number corresponding to OUTPUT DEVICE causes the displayed parameter to alternate (flip-flop) between CRT and PRINTER, where CRT indicates displayed output and PRINTER indicates output to the ISS start-up printer address.

With printed output, the entire Key File report is printed, followed by the reappearance of the KFAM-7 menu.

With displayed output, which is user-interactive, the SF keys listed in Table 6-3 are available. After accepting the displayed parameters, the KDR variables are displayed.

Refer to Appendix A if an error message appears.

Table 6-3. SF Keys for Print Key File

| SF KEY | DESCRIPTION |
|---|---|
| SF'3 | In the case of KIEs too long for one 64- or 80-character line, this key switches the displayed information to enable viewing of all information. |
| SF'4 | Displays the last physical KIR in the Key File. |
| SF'6 | Displays the KDR variables. |
| SF'7 | Displays the first physical KIR in the Key File. |

Table 6-3. SF Keys for Print Key File (continued)

| SF KEY | DESCRIPTION |
|---|---|
| SF'8 | Allows entry of the relative KIR to be displayed, where 0 (zero) displays the access tables, 1 displays the first KIR, 2 displays the second KIR, etc. |
| SF'9 | Redisplays the last item chosen. |
| SF'10 | Prints the last item chosen (current sector) to the ISS start-up printer address. |
| SF'11 | Moves the currently displayed sector five sectors forward from the previous sector location; the last KIR sector is redisplayed when reached. |
| SF'12 | Moves the currently displayed sector one sector forward from the previous sector location; the last KIR sector is redisplayed when reached. |
| SF'13 | Moves the currently displayed sector one sector backward from the previous sector location; the KDR variables are redisplayed when reached. |
| SF'14 | Moves the currently displayed sector five sectors backward from the previous sector location; the KDR variables are redisplayed when reached. |
| SF'20 | Displays the last KIE in the current KIR. |
| SF'23 | Displays the first KIE in the current KIR. |
| SF'25 | Redisplays the last item chosen. |
| SF'27 | Displays the next five KIEs within the current KIR. |
| SF'28 | Displays the next KIE within the current KIR. |

Table 6-3. SF Keys for Print Key File (continued)

| SF KEY | DESCRIPTION |
|---|---|
| SF'29 | Displays the previous KIE within the current KIR. |
| SF'30 | Displays the previous five KIEs within the current KIR. |
| SF'31 | Returns the KFAM-7 menu to the screen. |

## 6.5 REALLOCATE FILE SPACE

The Reallocate File Space utility and the ISS Copy/Verify utility can be used in conjunction with one another to lengthen or shorten KFAM Key Files and User Files. After one or more KFAM files have been copied using Copy/Verify, Reallocate File Space must be run for each KFAM file if the position of the END control sector has been changed in either the User File or Key File. See Chapter 5, Section 5.6 for KFAM file copy procedures. To prevent the utility from updating the recovery information and END control sector when a Secondary Key File is being accessed, a Key File Type option is available as described under the Operating Notes.

### Operating Notes

Default values are maintained for the User File name, address, and password (if any); the Key File number and address; and the Key File Type (Primary or Secondary). When the ENTER DESIRED FUNCTION prompt appears, entering the number corresponding to KEY FILETYPE causes the displayed parameter to alternate (flip-flop) between PRIMARY and SECONDARY. The KFAM-7 menu appears immediately upon completion of program execution.

## 6.6 REORGANIZE IN PLACE

Reorganize In Place is an operator-controlled utility chosen from the KFAM-7 menu, requiring operator-specification of file names, addresses and other parameters. Unlike Reorganize/Rebuild Subsystem which copies a KFAM User File and Key File, 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 all other records, until the entire User File has been placed in sequential order. During this process, all deleted records are removed. The Key File is then 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 work file, KFAMWORK, is required.

This program is four to five times slower than the Reorganize/Rebuild 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/Rebuild Subsystem.

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 10.5K partition if the record length exceeds 8 sectors.

CAUTION:

Before executing Reorganize In Place, backup copies of the User File and Key File must be made, since certain hardware or software errors, if encountered, destroy the KFAM file. Also, this utility cannot be used with duplicate keys or Secondary Key Files; use Reorganized/Rebuild Subsystem.

Operating Notes

Default values are maintained for the User File name, address, and password; and the Key File number and address. A prompt appears asking whether or not backup copies of the User File and Key File have been made; if not, it is recommended that the user touch SF'31 instead of entering a reply to that prompt. After completing reorganization, the KFAM-7 menu appears. If an error occurs, see Appendix A; it may be necessary to use the backup copies.

## 6.7 REORGANIZE/REBUILD SUBSYSTEM

Reorganize/Rebuild Subsystem performs the following KFAM file maintenance operations on a single Key File:

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. All output file records are reordered according to their ascending key values, and space previously occupied by deleted records is available following reorganization.

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.

4. Optionally, the utility will build (or rebuild) a Key File without reordering User File records. This option is especially useful for Secondary Key Files and is controlled by variable 06$ (line 4290 below).

Where multiple Key Files are used, one setup program must reorganize the User File according to the Primary Key File, and then rebuild the Primary Key File. Thereafter, one setup program is required to rebuild each Secondary Key File based up the reordered User File.

With the rebuild option, the Key File is rebuilt and the User File's data is not altered; however, a valid END control sector and valid recovery information (reflecting the Primary Key File) is required.

The modules (program files) that make up Reorganize/Rebuild Subsystem are as follows:

| File Name | Description |
|---|---|
| KFAMREFS | Copy/Verify Reference File |
| KFAM3507 | Start up, open files |
| KFAM3607 | Generate code |
| KFAM3707 | Reorganize, parts 1 and 2 |
| KFAM3807 | Build Key File |
| KFAM3907 | Reorganize, part 3, close files |
| KFAM2305, KFAM2325, KFAM2345, ISS2175 | Overlays |

Reorganize/Rebuild Subsystem modules may be copied to the desired disk(ette) using the supplied ISS reference file KFAMREFS during Copy/Verify operation. Specify INPUT MODE = INDIRECT to use a reference file during Copy/Verify operation.

## Writing The Setup Module

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

1. Lines 1-110 contain statements executed before Reorganize/Rebuild 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 and are executed after that module defines its common variables and sets default values. The variables and their assigned values are critical to the execution of this utility; line numbers need not be the same as listed below, but must be within the required ranges.

The master setup program is shown below. A line may be omitted if the default value shown is the desired value. A LINPUT or INPUT statement may be used if a parameter must be determined by the operator at run-time. Read all comments before writing a setup module.

| Line | Contents | Default Value | See Comment |
|---|---|---|---|
| 10 | REM program identification | | |
| 20 | PRINT HEX(03) | - | 1 |
| 50 | SELECT DISK (disk address for Reorganize/Rebuild 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 | O3$ = "C" catalog output User File if uncataloged.<br>= "Y" output User File already cataloged, do not catalog it.<br>= "N" output User File is not already cataloged, catalog it. | C | 9 |
| 4280 | C3 = number of sectors to allocate to output User File. This statement not needed if O3$ = "Y" (above).<br>*If the utility catalogs the file, the default value for C3 is the number of sectors in the input User File. | * | 9 |
| 4290 | O6$ = "C" to copy back output User File over input User file when reorganization completed.<br>= blank to leave input User File intact.<br>= "B" to rebuild Secondary Key File only. | blank | 10 |
| 4300 | N4 = output Key File number. | 1 | 11 |
| 4310 | P4$ = output Key File device address as "xyy". | | |

| Line | Contents | Default Value | See Comment |
|---|---|---|---|
| 4320 | 04$ = "C" catalog output Key File, if uncataloged.<br>= "Y" output Key File cataloged, do not catalog it.<br>= "N" output Key File not cataloged, catalog it. | C | 12 |
| 4330 | 04 = number of sectors to allocate to the output Key File. Omit this if 04$ = "Y".<br>*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.<br>= 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<br>*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 | | |
| 4410 | 07$ = either (1) write END control sector and recovery information; i.e., Primary Key File, if 07$=" " (blank), or (2) do not write the END control sector and recovery information; i.e., Secondary Key File, if 07$=" ". | blank | 15 |
| 4420 | 08$ = either (1) standard, unique keys if 08$= " " (blank), or (2) use duplicate key convention if 08$="D". | blank | 15 |

Comments

1. The CRT screen should be cleared prior to calling Reorganize/Rebuild 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 Reorganize/Rebuild 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 1-110 must be a LOAD DC that loads Module 1 of the utility and clears Lines 1-110 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 O3$, the output User File is cataloged by this utility if it does not already exist. "C" is the default value of variable O3$ (line 4270).

   If "N" is assigned to variable O3$, the system ensures that the named output User File does not already exist on the disk and then 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 O3.

   If "Y" is assigned to variable O3$, the system checks that the named output file already exists. Variable O3 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), 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, the output Key File name is constructed from the output User File name and the output file is not copied back. If variable 06$="B", the User File is not copied but the Secondary Key File specified is rebuilt according to the reordered output User File. The Primary Key File must be reorganized first with 06$="C" or 06$=(blank) and then each Secondary Key File rebuilt with 06$="B".

    If 06$="B", the output file variables need not be included, since there is no output file. Also see Comment 15.

11. This number becomes the 6th character in the construction of the output Key File name. 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 P9$ is blank, 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 the assigned password must be provided for any station to open that file.

15. The parameters for variables 06$, 07$, and 08$ are critical when multiple Key Files are used. Although Reorganize In Place is allowed to reorganize and rebuild the Primary Key File according to reordered User File record sequence, use of Reorganize/Rebuild Subsystem is required following Key File Recovery and is recommended unless insufficient disk space exists to run Reorganize/Rebuild Subsystem.

    The typical procedure is to run Reorganize/Rebuild Subsystem (or Reorganize In Place, if allowed) for the Primary Key File first, with 06$="C" or " " (blank), 07$=" " (blank), and 08$=" " (blank). Each Secondary Key File can then be rebuilt based on the reordered User File records, with the reorganized User File specified as the input User File (lines 4210 and 4220), 06$="B", 07$="S", and either 08$="D" if keys have been defined as duplicate or 08$" " (blank) if keys are unique.

Shown below is an example of a KFAM-7 setup program.

```
10 REM EXAMPLE OF A REORGANIZATION SETUP PROGRAM
20      PRINT HEX(030A0A0A0A); "REORGANIZE INVENTORY FILE"
50      SELECT DISK 310 :REM REORGANIZE/REBUILD 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    O6$ = "C" :REM COPY BACK OUTPUT USER FILE
4310    P4$ = "B20"
4340    N5$ = "START"
4350    P5$ = "310"
```

<u>Reorganize/Rebuild Subsystem Operation</u>

The operation of Reorganize/Rebuild 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, using a special procedure, based on the keys in the output User File. The new Key File, optionally, may occupy the same physical space as the old Key File.

3. If indicated by the setup program, the output User File is copied back to the input User File, overwriting the original.

4. With the rebuild option (O6$="B"), instead of the preceding steps, the input User File is read and the Key File is built.

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 and output User Files are intact, and a Key File may be built for either one, using the Key File Recovery utility. During Part 3, both the output User File and the Key File remain intact. 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 no operator intervention is normally required. However, there are recovery procedures for certain error conditions. These are described in Appendix A.

## 6.8 CONVERT TO KFAM-7

Since the Key File structure employed for KFAM-3 and -4 differs from the KFAM-7 Key File structure, the Convert to KFAM-7 utility is provided to convert KFAM-3 or -4 files into KFAM-7 format. KFAM-5 and -7 Key Files are identical.

Backup copies of the User File and Key File must be made prior to conversion, and both the User File and Key File must be on-line during program execution. After conversion, the file may be accessed by KFAM-7 software; Initialize KFAM File and Build Key File are not needed.

CAUTION:

Make backup copies of the KFAM-3 or -4 User File and Key File prior to executing this program. Certain errors can destroy the file being converted.

### Operating Notes

Default values are maintained for the User File name, address, and password; the Key File number and address; and the input file type. The input file type alternates (flip-flops) between KFAM-3 and KFAM-4 when the appropriate number is entered in reply to the ENTER DESIRED FUNCTION prompt.

The KFAM-7 menu appears upon completion. Should an error message appear, refer to Appendix A.

## 6.9 BUILD SUBROUTINE MODULE

Build Subroutine Module creates a module (program file) containing the KFAM subroutines chosen by the programmer. It allows the programmer to include in the module only those subroutines and subroutine capabilities actually needed, thereby minimizing the amount of memory required for KFAM subroutines. For each KFAM-7 version, a subset of subroutines may be saved to disk and subsequently used, as an alternative to the complete set of subroutines contained in program files KFAM0107 (Single Bank version), KFAM0207 (Multiplexed verions), or KFAM0307 (Multiple Bank version).

The module containing the subroutines can be assigned any file name and saved at any ISS start-up disk address. However, if a file (active or scratched, data or program) of the same name exists at the specified address, it will be overwritten (replaced) with the chosen subroutines. If extra sectors are desired for the subroutine module, it may be cataloged in advance as a data file.

## Operating Notes

Default values are maintained for the output file name and address, and the KFAM-7 version desired. The output file name and output file address determine whether a new file is created or whether an existing file is reused and possibly converted to an active program file.

After accepting the displayed values, a list of subroutines and options appear. Each displayed subroutine may be chosen by touching the corresponding SF key, which causes an asterisk (*) to appear to the left of its SF key number and name. All KFAM-7 subroutines are functionally described in Table 5-1, while Table 6-4 lists the SF keys available during Build Subroutine Module operation.

Please note the operational function performed by SF'29, SF'30, and SF'31. After SF'30 is touched, PHASE 2 - BUILDING MODULE appears briefly and is soon replaced by the KFAM-7 menu.

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 bank at any one time.

Table 6-4. Build Subroutine Module Options

| SF KEY/SUBROUTINE | DESCRIPTION |
|---|---|
| SF'01 through SF'10 | KFAM subroutines (see Table 5-1). |
| SF'11/239 CLOSE | KFAM Close subroutine which writes neither recovery information nor the END control sector necessary for Key File Recovery (and the rebuild option of Reorganize/ Rebuild Subsystem). Usually only chosen if records will not be added or deleted. |
| SF'12/CLOSE WITH RECOVERY INFO | KFAM Close subroutine which writes neither recovery information (KDR) nor the END control sector in the User File, which is necessary for Key File Recovery. |
| SF'13 and SF'14 | KFAM subroutines |
| SF'15/217 MUX OPEN | For non-KFAM files (sequential files), this subroutine opens a file if it is cataloged. |
| SF'16/ MUX OPEN NEW | For non-KFAM files (sequential files), this subroutine opens and catalogs (creates) the file. |
| SF'17/218 MUX END | For non-KFAM files, this subroutine writes an END control sector. |
| SF'18/219 MUX CLOSE | For non-KFAM files, this subroutine closes the file. |
| SF'29 | Cancels the subroutines previously chosen (erases all asterisks). |
| SF'30 | Processes the chosen subroutines and saves them as a disk program file. |
| SF'31 | Aborts this utility and returns the KFAM-7 menu to the screen. |

## 6.10 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 operating on the file adhere to the following conventions:

1. All deleted records must be flagged in the User File with HEX(FF) in the first byte of the key. Otherwise, both active and deleted records will be included in the Key File, necessitating the deletion of unwanted keys.

2. With KFAM-7, CLOSE is a system requirement. If Build Subroutine Module was used and the Close with Recovery Information option was <u>not</u> selected, the user must include the Write Recovery Information subroutine and must execute the Write Recovery Information 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 multiple Key Files existed for the User File, the following procedures are recommended:

1. If the Primary Key File was destroyed, run the Key File Recovery utility to recatalog and rebuild the Primary Key File.

2. For each destroyed Secondary Key File, run the Initialize KFAM File utility.

3. Run the Reorganize/Rebuild Subsystem first to reorganize and rebuild the Primary Key File. Then run Reorganize/Rebuild Subsystem to rebuild each destroyed Key File.

### Operating Notes

Default values are maintained for the User File name, address, and password (if any); the Key File number and address; key type; and whether the Key File is cataloged. The Key File option alternates (flip-flops) between CATALOGED and NOT CATALOGED when the appropriate number is chosen in reply to the ENTER DESIRED FUNCTION prompt. Similarly, the Key Type option alternates (flip-flops) between STANDARD (unique keys) and DUPLICATE (duplicate keys).

Upon successful completion of utility execution, the KFAM-7 menu appears. Error messages include a list of duplicate keys and are described in Appendix A.

## 6.11 RESET ACCESS TABLES

An entry is maintained in the User File's access table for each possible 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; if the station is not currently accessing the file, a blank appears. Other access tables are maintained in the KDR and in global memory, but these depend upon the KFAM-7 version currently in use. The User File and KDR access tables may be printed by the Print Key File utility for all KFAM-7 versions.

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, one can determine which station has the file open in the access mode that conflicts with the desired access mode. 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; this must be corrected by running Reset Access Table.

Other information is also cleared, depending upon the KFAM-7 version in use. With the Multiplexed version, the KDR, program hog (global variable @T), and the User File access table are cleared. With the Single Bank and Multibank versions, the table of open files (global array variable @T$()), the queue (global variables @Q$ and @Q9$), protected sectors (global array variable @V4$()), program hog (global variable @T), and the User File access table are cleared.

Access table information can be cleared for all stations (1-16) or a specified station number. All files listed in the global table of open files may be cleared with the Single Bank or Multiple Bank versions, or specified files may be cleared with all versions. The work file, KFAMWORK, located at the ISS loading address is automatically cleared if all files are to be cleared, and it may be one of the specified files to be cleared at any address.

CAUTION:

This utility should be used with extreme caution. When a specified station number is used, make sure the station is no longer accessing KFAM files before using this utility. If all station numbers are specified, be sure that <u>all</u> stations are no longer accessing KFAM files before using this utility. Reset Access Tables indiscriminately sets all access table information blanks, whether the entries are valid or the result of an accident.

## Operating Notes

Default values are maintained for the mode and station number. The mode indicates whether all or specified files are to be reset. Consistent with its use with the ISS utilities, the mode is specified as one of the following:

1. If the mode is ALL, all files listed as open in global memory are reset (valid only for the Single Bank and Multiple Bank versions).

2. If the mode is PART, specified files are reset. After accepting the displayed values, the User File name, the disk addresses of the Key File and (for the Multiplexed version only) the User File (xyy/xyy form), and the Key File number are requested for each KFAM User File/Key File to be reset (or KFAMWORK). Enter 0 (zero) instead of a file name to indicate completion.

3. If the mode is INDIRECT, the files specified by means of a reference file are reset. The ISS utility, Create Reference File, allows a reference file to be created, edited, or printed. The reference file must have been created using Create Reference File (see Chapter 2, Section 2.3) prior to running Reset Access Tables, and the entries must be in the same form (filename, xyy/xyy, n) as in the PART mode. During Reset Access Tables, only the reference file name and its disk address need to be specified, thereby eliminating operator entry of each file's parameters.

The station number can be specified as 0 (zero) or any number from 1 through 16. If 0 (zero) is specified, access tables are reset for all station numbers; otherwise, only the access tables for the specified station number are reset.

Following completion of utility execution, the KFAM-7 menu appears. Error messages are listed and described in Appendix A.

# CHAPTER 7
# THE KFAM-7 SUBROUTINES

## 7.1 INTRODUCTION

The KFAM subroutines simplify KFAM file access and perform the maintenance operations most frequently performed on files organized by KFAM. 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.

Each KFAM subroutine is marked with a DEFFN' statement and performs a unique function. User-supplied programs call a KFAM subroutine via a GOSUB' statement. Certain parameters which exactly define the function to be performed are included in the GOSUB' statement in an argument list passed to the subroutine. After completing the prescribed task, the subroutine provides a return code in KFAM user variable Q$, which indicates whether or not the subroutine was executed successfully and if a special condition was encountered. Certain subroutines also set other KFAM user variables.

### Reserved Variables and DEFFN' Statements

The KFAM subroutines collectively maintain control information in memory-resident variables in conjunction with disk-resident control information stored mostly in the Key File. It is extremely important that the values of the KFAM variables never be altered by a user-supplied program, except for the special cases discussed in Chapter 8 of this manual. Compatible with the general convention of reserving all variables and arrays in the range of Q through W for ISS-5 software (with the exception of SORT-4), KFAM subroutines use variables and arrays (alpha and numeric) whose first letter begins with Q, R, T, or V, including global variables and arrays @Q, @T, and @V. These variables are reserved for KFAM-7 use and must be handled as Read Only variables by user-supplied software.

Similarly, the ISS-5 convention of reserving DEFFN' statements from DEFFN'200 through DEFFN'255 for ISS software applies to KFAM-7, which reserves DEFFN'212 through '219 and DEFFN'230 through '239 (inclusive) for KFAM-7 subroutines. The need for user-supplied software to observe this convention is especially important with global 2200MVP operation, where the calling partition is searched for the corresponding DEFFN' statement before the currently selected global partition is searched for the corresponding DEFFN' statement.

## Identification of KFAM Files

According to 2200 device table characteristics, each User File and each Key File are identified by one slot, or file number, from #0 through #15. The file number and its address are entered into the device table via a SELECT statement within the Open subroutine for both the User File and Key File. The User File number is specified in the DATALOAD and DATASAVE statements incorporated into user-supplied software, whereas the Key File number is used exclusively by KFAM subroutines.

Since a User File and Key File must be open simultaneously for indexed record access to occur, a User File and its Key File are considered by KFAM to be one KFAM file. The KFAM OPEN subroutine argument list includes the User File number (used by KFAM record access subroutines to set the Current Sector Address), the Key File number (used by most KFAM subroutines), and a unique digit from 1 through 8, which uniquely identifies this User File/Key File; this number is referred to as the KFAM ID number.

Since the KFAM ID number is specified as an OPEN subroutine argument, subsequent references to KFAM subroutines need only include the KFAM ID number, and not the User File's name, file number, disk address, and similar Key File parameters.

If multiple Key Files are used to provide alternate key paths to the same User File, each User File/Key File pair requires a different KFAM ID number, the User File must have different file numbers for each User File/Key File pair with DC mode access, each Key File must have different file numbers, and other special considerations must be observed (see Section 7.4).

## KFAM Subroutine Calling Sequences and DATALOAD/DATASAVE Statement Use

As described in Table 5-1, there are the following groups of KFAM-7 subroutines:

- The General Purpose subroutines are OPEN, CLOSE, RE-OPEN, and WRITE RECOVERY INFORMATION.
- The Key Sequence Access subroutines are FINDFIRST, FINDNEXT, FINDLAST, and FINDPREVIOUS.
- The Random Access subroutine is FINDOLD.
- The Add and Delete subroutines are FINDNEW, FINDNEW(HERE), and DELETE.
- The Special Purpose subroutine is RELEASE.

For each KFAM file, the OPEN subroutine must be the first KFAM subroutine called. All other subroutines require that the specified KFAM file is already open. After access operations on a KFAM file are complete, the CLOSE subroutine should be called promptly, since other stations may be waiting to access that KFAM file. Once closed, that KFAM file can no longer be accessed until the OPEN subroutine is called again.

The General Purpose and Special Purpose subroutines are called without any associated DATALOAD or DATASAVE statements. These subroutines include OPEN, CLOSE, RE-OPEN, WRITE RECOVERY INFORMATION, and RELEASE, none of which pertain directly to record access.

The Key Sequence Access and Random Access subroutines provide record access functions, as denoted by the prefix "FIND" in their names. Following each successful record access subroutine call, KFAM resets that station's Current Sector Address (for DC mode access), equates the KFAM variable T6 to the relative sector address (for DA or BA mode access), and returns the record number for blocked records in the KFAM variable Q. (See "Programming Considerations for BA and DA Mode Access" for more information.) With this information, the user-supplied program executes a DATALOAD statement to read the appropriate User File record. When records are to be updated, certain considerations must be observed, as discussed later in this section. Record access subroutines include FINDFIRST, FINDNEXT, FINDLAST, FINDPREVIOUS, and FINDOLD.

The Add and Delete subroutines are used to perform the following functions: (1) add a new User File record, (2) delete a User File record no longer needed, and (3) change the key of an existing User File record. The sequence in which these subroutines are used along with DATALOAD and DATASAVE statements depends upon the function to be performed. Subroutines in this group include FINDNEW, FINDNEW(HERE), and DELETE.

To add a new User File record, the FINDNEW subroutine is called, followed by a DATASAVE statement. The FINDNEW subroutine locates the key in the Key File and sets the Current Sector Address (DC mode) and variable T6 (BA, DA modes); the DATASAVE statement writes the record at the appropriate position. With blocked records, special considerations are required (see "Programming Considerations for Blocked Records").

To delete a User File record, the DELETE subroutine is called to remove the key from the list of KIEs in the Key File and sets the Current Sector Address and KFAM variable T6 to allow User File access. The record should be flagged as deleted with a hexadecimal (HEX) FF value in the first byte of its key. The deleted record is typically read using a DATALOAD statement, its key's first byte is changed to HEX(FF), and the record is written into the same sector using a DATASAVE statement. With DC mode access, a DBACKSPACE statement is required between the DATALOAD and DATASAVE statements to offset the automatic incrementing of the Current Sector Address by the DATALOAD statement.

To update a record's key, the following subroutine and statement sequence is required:

1. A FINDOLD subroutine is optionally executed to ensure the new key does not already exist in the Key File. If found, this sequence is aborted.

2. A DELETE subroutine is executed to remove the old key from the Key File, and sets the Current Sector Address and KFAM variable T6 to the record location.

3. A FINDNEW(HERE) subroutine is executed to add the new key to the Key File, without altering the User File record's address.

4. The record is read using a DATALOAD statement, the key value is changed, and a DATASAVE is executed to save the updated record. With DC mode access, a DBACKSPACE statement is executed between the DATALOAD and DATASAVE statements to offset the automatic incrementing of the Current Sector Address by the DATALOAD statement.

To update a record without altering its key, any record access subroutine may be used to locate the appropriate record, and step 4 is performed except that any record's key may not be changed.

## Programming Considerations for Blocked Records

When a file contains blocked records (multiple records per sector), special precautions are required whenever records are to be written, since a DATASAVE statement writes over the entire sector, any valid data record already written into that sector would be irretrievably lost.

In order to save two or more records into the same sector, the argument lists for those records must be strung together in a single DATASAVE statement argument list. A DATALOAD statement with the same argument list can be used to read the variables back into memory. A drawback of such an arrangement is that it forces the programmer to work with all the records in a sector when records are to be written.

When a KFAM record access subroutine is called, in addition to setting the User File Current Sector Address to the appropriate sector (for DC mode access) and setting KFAM variable T6 to the relative sector address (for DA, BA mode access), KFAM accommodates blocked records by returning a variable Q, whose value indicates the record within the sector to be processed. The value of variable Q must never be changed by user-supplied software, but it can be used as a subscript to determine those array elements containing the desired record, for instance, A$(Q), B$(Q), and N(Q). With nonarray, contiguous records, variable Q may be used to calculate the relative byte location of the appropriate record.

Reading the file presents no problem since the programmer can use any of the multiple records and ignore the others. To read each record, (1) the appropriate KFAM record access subroutine is called, and (2) a DATALOAD statement is executed.

However, writing one record requires that the entire sector be written due to the 2200 disk data handling characteristics. A record may be written either as a new data record added to the file or as an existing data record whose contents have been updated.

When adding new records to a KFAM file, a new key is first added to the Key File via a FINDNEW subroutine call. KFAM assigns each station number executing a FINDNEW subroutine to a different sector, each of which contains enough space for at least one record. Within each sector, KFAM assigns record numbers (the variable Q) in ascending order beginning with 1. Only if FINDNEW returns a value of Q=1, the programmer need not be concerned about overwriting active records in that sector (since there are none); however, all inactive records should be flagged with a hexadecimal FF in the first byte of the key.

The steps involved in writing new records into a file are as follows:

1. The FINDNEW subroutine is called, which adds a key to the Key File. KFAM resets the User File Current Sector Address (and variable T6) and returns the record number in variable Q.

2. If Q=1, the program need not read the sector; skip to Step 6.

3. To prevent overwriting valid records, a DATALOAD statement is executed which reads all possible records from the sector into its argument list. These records are saved along with the new record when it is written.

4. Following the DATALOAD statement, the User File Current Sector Address is automatically incremented by the system. With DC mode access, a DBACKSPACE statement is executed to position the User File Current Sector Address to the previous sector location.

5. The record to be added is equated to the appropriate array elements by using variable Q as a subscript; for instance, A$=A$(Q): B$=B$(Q): N=N(Q).

6. All inactive records must be flagged by equating the first byte of each inactive record's key to HEX(FF).

7. The entire sector is then saved by a DATASAVE statement.

8. The record's key has been added to the Key File (Step 1), and the User File now contains the record at the appropriate User File location (Step 7). For an example of this procedure, refer to the description of the FINDNEW subroutine in Section 7.12.

When updating blocked records, a similar procedure is necessary. A KFAM record access subroutine (e.g., FINDOLD) is called, which sets the User File Current Sector Address (and KFAM variable T6) and returns the record number in variable Q. Unlike adding new records, the sector containing the record to be updated always contains active records. Therefore, the entire sector is always read using a DATALOAD statement, and a DBACKSPACE statement is then executed with DC mode access. The appropriate array elements are updated and the entire sector is then resaved using a DATASAVE statement. Care must be taken to change only those variables which correspond to the record to be updated.

## Programming Considerations for BA and DA Mode Access

Because absolute sector addressing is used with BA and DA mode access, 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 a KFAM subroutine, variable T6, the sum provides the absolute sector address necessary for the x value of the DATALOAD BA statement 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 returned in variables T6 and T4$ are not defined. (T4$ is simply the value of T6 and Q in hexadecimal (hex) form; however, the value of B must be converted to hex form before adding T4$ and B, to produce a hex value of x.)

## 7.2 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 7-1 with a description of record access conventions to be followed, as well as those file access modes allowed and not allowed by other stations attempting to access the file. Under the Record Access Description column in Table 7-1, "read" and "write" apply to both the User File and the Key File. It is the user's responsibility not to update a file opened in the Inquiry or Read Only access modes.

Table 7-1. 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:

Build Key File
Reallocate File Space
Reorganize In Place
Reorganize/Rebuild Subsystem
Convert to KFAM-7
Reset Access Tables
Key File Recovery

Print Key File uses the Read Only access mode. Build Subroutine Module hogs the disk during its execution.

## 7.3 GENERAL PROGRAMMING REQUIREMENTS

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

1. Each KFAM-7 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). The KFAM-7 variables in KFAM0007 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 Chapter 5, Section 5.7.

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 are thus 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.

With nonglobal application programs, both variables and KFAM subroutines are contained in the application program.

2. Before opening any KFAM file, the variable S2 must be equated to the station 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 global KFAM-7 subroutine, the appropriate global subroutine modules must be loaded and run (see Chapter 5, Section 5.7), and 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 (e.g., to reference global variables stored in the Multibank version's KFAMCOM global partition), 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).

   It is recommended that during program initialization, the user-supplied program check if the global partition KFAM is running before continuing program execution, for instance, via:

   ```
   50 SELECT @PART "KFAM": ERROR $BREAK: GO TO 50
   ```

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

5. The user-supplied program should provide an error recovery procedure that will close all open KFAM files 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 thereby avoid the need to close KFAM files (especially for "background" user partitions).

6. In the Shared access mode, 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 DATASAVE be executed; therefore, the protect flag should be set for all of these operations. Operations which only execute DATALOAD on the record should not set record protection.

7. A DATASAVE DC END statement must not be used to write the END control sector (following the last sector of live data records). KFAM maintains the END control sector's position automatically in the Key File's KDR and updates the END control sector's position in the User File when the file is closed. A DATASAVE DC END would destroy the User File access table and password. User-supplied programs must <u>never</u> write trailer records of <u>any kind</u> into the User File.

8. The KFAM 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, if running in a 2200MVP, 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
```

   For 2200VP-resident programs, a delayed retry is recommended.

9. User-supplied programs must never make any assumptions about the status of User File sectors other than those specifically returned by a subroutine. For example, 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.

10. 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 subroutines can search the Key File without concern for protected records.

c. As an alternative to disk hog mode use.

11. If the previous subroutine call set the protect flag and there may be a long delay before the next KFAM 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 or RETURN CLEAR 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 file. A Special Function key subroutine might be made available to close a file at any time. The operator should always have a procedure for closing the file available 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 Tables utility must be run.)

    Similarly, it is recommended that KFAM subroutine calls involving FINDNEW, FINDNEW(HERE), and DELETE immediately precede their associated DATALOAD and/or DATASAVE statements. By reducing the time in which an accident could occur; the probability of User File/Key File incongruence or losing a record due to error or power failure is also reduced.

13. The Multiplexed KFAM-7 version allows the option of hogging the current key file's disk drive for this station. KFAM also maintains a "Key File busy/free" flag in the KDR which ensures that only one KFAM subroutine will be executed on any one Key File at any time.

    If the programmer specifies the Key File disk drive hog, an additional return code of "X" may indicate that, if the Key File hog was requested, the Key File is currently busy (hogged) due to another station. In this case, either a retry without hogging requested or a delayed retry with hogging requested after at least a 200 ms. delay is recommended.

14. After calling any KFAM subroutine, the value of KFAM variable Q$ must be checked upon return. If Q$ does not return a blank, unsuccessful subroutine execution occurred, or a special condition was encountered, and the appropriate error or special condition routine should be accessed. If Q$="B" on a record access subroutine, a protected (busy) sector was encountered, and a delayed retry is recommended. For other values of Q$, line numbers or other means of isolating the probable cause might be displayed.

## Using GOSUB' Statements and Argument Lists

The DEFFN' statement which marks each KFAM subroutine requires certain parameter values to be passed from the GOSUB' statement which calls it. Passed parameter values 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 element 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 and their meanings are described in Table 7-2.

Table 7-2. KFAM Subroutine Argument Symbolic Variables

| SYMBOLIC VARIABLE | MEANING |
|---|---|
| I | KFAM ID number (usually 1-8, although CLOSE requires a negative KFAM ID number for Secondary Key Files, -1 through -8). Each KFAM User File/Key File open must be assigned a unique KFAM ID number, which is a required argument for most subroutines. |
| K | Device table file number assigned to the Key File. |
| U | Device table file number assigned to the User File. |
| F | Key File number (usually 1-9, although -1 through -9 is valid for Secondary Key Files) specified as the 6th character in the Key File name (usually assigned during Initialize KFAM File). |
| A$ | The record's key (alphanumeric). |
| N$ | User File name. |
| A | Access mode as described in Table 7-1 (1-4). |
| P$ | File password, if any (usually assigned 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. With the Multiplexed KFAM-7 version, P=2 or P=3 indicates hold hog mode of the Key File's disk drive. |
| "xxx" | Device address, Key File. |
| "yyy" | Device address, User File. |

## Return Codes Q and Q$

Upon returning to the user 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 7-3.

Table 7-3. 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), or beginning of file (FINDPREVIOUS). |
| "M" | On an Open, the User File is not a Multiplexed/Multistation File. |
| "N" | Key not found (FINDOLD). 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.)<br><br>For an OPEN, this indicates too many files open to accommodate opening this file (global memory tables are full). |

Table 7-3. KFAM Subroutine Q$ Return Codes (continued)

| Q$ VALUE | MEANING |
|---|---|
| "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 7-1. |
| "P" | Invalid password. |
| "X" | Improper call to a KFAM subroutine, (argument values erroneous, etc.). Also, if the Key File hog was requested, the Key File is currently hogged by another station. A retry with hog requested after a 200 ms. wait is recommended, or an immediate retry without hog requested may be performed. |

If Q$ is anything other than blank, the User File Current Sector Address parameter and KFAM variables T6 and Q are undefined. Immediately upon return from any of the subroutines, the user-supplied program should check Q$ for possible error indications. For instance, Q$ " " indicates an error or special condition.

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 ID number not an integer between 1 and 8. | X<br>ERR P34 |
| KFAM ID number is the same as ID number for a file already open. | X |

| Error | Q$ Value or ERR Code |
|---|---|
| File to be opened is already open. | X |
| Individual file numbers not integers between 0 and 15. | ERR P57<br>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<br>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<br>ERR D84 |

## 7.4 GUIDELINES FOR MULTIPLE KEY FILE AND DUPLICATE KEY USE

Multiple Key Files can be maintained by user-supplied programs for a single User File, if certain guidelines are observed. If different Key Files index different key locations within each User File record, multiple key paths can be used to access the User File records; for instance, in ascending key sequence or by whichever key is known.

NOTE:

Multiple Key File use should be attempted only by advanced programmers who have prior experience using KFAM-7. Others should skip to Section 7.5. Advanced programmers investigating multiple Key File and duplicate key use should carefully read the information in this section before designing a multiple Key File application program. Refer to Table 6-1 before using a KFAM utility on a Secondary Key File.

## KFAM-7 Provisions for Multiple Key Files and Duplicate Keys

KFAM provides for multiple Key Files in the following ways:

1. The Initialize KFAM File utility and other KFAM-7 utilities allow a Key File number from 1 through 9 to be specified. Typically, only Key File number 1 is used unless multiple Key Files are present.

2. The KFAM-7 OPEN subroutine allows the same User File to be reopened for multiple Key Files. The first Key File, the Primary Key File, is opened along with the User File. For each subsequent Key File to be open (each Secondary Key File), the OPEN subroutine argument for the Key File number must be a negative digit, -1 through -9, to indicate that the User File is to be reopened while opening the specified Key File, as further described in Section 7.5. The absolute value of Key File numbers used for the same User File must be unique.

3. The KFAM-7 CLOSE subroutine allows a Key File (usually a Secondary Key File) to be closed without closing the User File and writing the END control sector and recovery information into the User File. This option is specified by a negative KFAM ID number in the CLOSE argument list. To close the Key File whose recovery information is most important (usually the Primary Key File but never a Key File to which new records have been added using FINDNEW(HERE)), a positive KFAM ID number is used in the CLOSE argument list, which writes the END control sector and recovery information into the User File, and closes that Key File and the User File.

4. Sector (record) protection is supported for multiple Key Files with the Single and Multiple Bank versions only.

5. The KFAM utilities Build Key File, Reorganize/Rebuild Subsystem, and Key File Recovery support duplicate keys, as described in Chapter 6. Other utilities allow entry of a Key File number 1-9.

## Functions Not Performed by KFAM-7 for Multiple Key Files and Duplicate Keys

The user who desires to maintain multiple Key Files must be aware of the functions KFAM-7 does not perform and provide these functions in user programs. Each User File/Key File pair requires a unique KFAM ID number. Each Key File must have a unique file number; the User File must have a unique file number for each KFAM ID number.

KFAM maintains separate global access table entries for each open Key File, and each Key File is maintained independently of other Key Files, regardless of whether they index the same User File.

KFAM-7 does not provide the following functions necessary for multiple Key Files:

1. All Key Files are not updated simultaneously. Since each Key File is independent of others and KFAM does not provide cross-Key File updates, the user's program must update multiple Key Files by calling a subroutine once for each Key File.

2. With the Multiplexed KFAM-7 version, sector (record) protection is maintained independently for each Key File. The user's program must employ its own record protection technique and might open the files in the Exclusive mode to perform updates.

3. Recovery information can only be stored for one Key File. The Key File closed with a positive KFAM ID number has its recovery information stored in the User File by overwriting previously saved recovery information; this is typically the Primary Key File.

4. After reorganizing the User File in primary key sequence or after running the Key File Recovery utility, the Secondary Key Files must be rebuilt using Reorganize/Rebuild Subsystem.

## Opening Multiple KFAM Files

In a multiple Key File environment, the Primary Key File must be opened first. A positive (normal) Key File number is specified in the OPEN argument list for the Primary Key File.

For each Secondary Key File, OPEN is called with a negative Key File number argument after the Primary Key File has been opened. The negative Key File number causes the User File to be reopened (instead of opened).

An example of opening multiple Key Files follow:

```
100  GOSUB'230 (1,1,2,1,"KFAMF010",3,"D10","D10"):REM PRIMARY
110  GOSUB'230 (2,3,4,-2,"KFAMF010",3,"D10","D10"):REM SECONDARY
```

## Adding Records

In order to preserve the User File/Key File congruence needed to allow access to User File records via multiple key paths in an on-line, real-time environment, when a record is to be added to the User File, the keys corresponding to each Key File must be added via KFAM subroutine calls. If a secondary key which is not necessarily unique, the user must create the unique key using the KFAM-7 key/pointer concatenation convention.

The primary key must first be added to the Primary Key File using the FINDNEW subroutine. Upon return, the pointer (KFAM variable T4$) contains the appropriate record location information for key/pointer concatenation. A FINDNEW(HERE) call for each Secondary Key File is executed. If duplicate keys are allowed, the keys must first be concatenated with the variable T4$. Use of the FINDNEW subroutine for Secondary Key Files is not recommended, since using FINDNEW(HERE) adds the key to the Secondary Key File and uses the sector and record number preset in variable T4$; with FINDNEW, however, the sector and record number returned by FINDNEW must be checked against the address and record location returned in variable T4$ for the previous FINDNEW for the Primary Key File. Using FINDNEW(HERE) is therefore recommended for adding KIEs to Secondary Key Files. The User File record may be written after all Key Files have been updated.

An example of adding a record to multiple Key Files follows. The following variables are used: K1$ is the primary key, K2$ is a secondary key, T4 is a KFAM variable for key length, T4$ is the KFAM variable for sector address and record location (the pointer), and T5$() is an array containing several KFAM variables where the array element is the KFAM ID number. At the location STR(T5$(n),1,3), is the variable T4$ for the file with the KFAM ID number of n, which must be preset before calling FINDNEW(HERE) for the Secondary Key File. Statement line 220 below is only required if duplicate keys are specified; this line uses the key length variable L as defined by the user (without the three-byte printer).

```
200  GOSUB'233(1,1,K1$,0):REM FINDNEW, PRIMARY KEY FILE
210  REM SAVE RECORD USING DATASAVE
220  STR(K2$,L+1,3)=T4$:REM SETUP UNIQUE KEY, K2$
230  STR(T5$(2),1,3)=T4$:REM PREPARE SECOND FILE
240  GOSUB'234(2,1,K2$,0):REM FINDNEW(HERE), SECONDARY KEY FILE
```

## Deleting Records

Deleting records requires that DELETE be called first for the Primary Key File, and then called for each Secondary Key File. After calling DELETE for all Key Files, the deleted record should be flagged with HEX(FF) in the first byte of each key (or the user may flag the entire record as ALL(HEX(FF)) using the ALL function). All variables are KFAM variables except variable L, the key length (not including the three-byte pointer).

```
300  GOSUB'231(1,1,K1$):REM DELETE, PRIMARY KEY FILE
310  STR(K2$,L+1,3)=T4$:REM SETUP UNIQUE KEY, K2$
320  GOSUB'231(2,1,K2$):REM DELETE, SECONDARY KEY FILE
```

## Locating Random Records

If the primary key associated with a record is known, the record location can easily be obtained by calling the FINDOLD subroutine. If only a secondary key is known and it is unique, the record location may easily be returned by calling FINDOLD. However, if the secondary key is a duplicate key, the lowest possible pointer (HEX(000000)) must be concatenated onto the secondary key before FINDOLD is called. FINDNEXT returns the position of the secondary key with the lowest pointer value. The record may be read using a DATALOAD statement. If it is not the desired record, FINDNEXT can be repeatedly called and read to locate the next highest key value. When the key value contained in KFAM variable T7$ changes (excluding the last three bytes), indicating a new secondary key value has been encountered and the record just located need not be read.

```
400 STR(K2$,L+1,3)=HWX (000000) :REM LOWEST POINTER
410 GOSUB'232 (2,1,K2$) :REM FINDOLD (SECONDARY)
420 IF Q$="X" OR Q$="B" THEN 600 :REM TEST RETURN CODE
430 GOSUB'237(2,1) :REM FINDNEXT (SECONDARY)
440 IF Q$="E" THEN 490 :REM TEST FOR LAST RECORD
450 IF Q$="X" OR Q$="B" THEN 620 :REM TEST RETURN CODE
460 IF STR (K2$,1,L) <> STR (T7$,1,L) THEN 490 :REM TEST IF NEW KEY
470 REM PROCESS RECORD
480 GOTO 430 :REM FIND NEXT RECORD
490 REM PERFORM NEXT TASK, SUCH AS INPUT NEXT KEY
.
.
.
600 IF Q$ <> "B" THEN 640 :REM TEST IF BUSY
610 $BREAK 5: GOTO 410 :REM BUSY, RETRY
620 IF Q$ <> "B" THEN 640
630 $BREAK 5: GOTO 430
640 REM ERROR HANDLING.  SIGNAL, ERROR, POSSIBLY CLOSE FILES
```

## Sector Protection with Multiple Key Files

Sector protection with multiple Key Files allows only the same station number protecting a sector to access that sector, regardless of whether the sector is protected for one or more KFAM files (KFAM ID number) by that station. To release sector protection, the RELEASE subroutine should be called once for each KFAM ID number accessing that sector.

## Recommendation for Protecting Sectors with the Multiplexed KFAM-7 Version

Since the Multiplexed KFAM-7 version does not support sector protection when multiple Key Files are used, the user must implement a sector protection scheme. A list of protected sectors in a disk file can be maintained, or a flag indicating whether that record is protected might be provided for in each User File record. Before the sector is protected or while the flag is being set, either the disk or program execution (2200MVP only) should be hogged. Alternatively, updates might be performed using the Exclusive access mode.

## Other Record Access Operations

Record access operations, such as reading a file in ascending logical key sequence, require that only the Key File whose key path is to be used be opened. Any operations involving updating records may require DELETE and FINDNEW(HERE) calls for each Key File whose key was changed when the record was updated. As with any KFAM file, any operations affecting the User File/Key File congruence require special considerations to preserve the congruence, as noted in this section.

### Closing Multiple KFAM Files

Since there is only one next-to-last sector in the single User File, Key File recovery information may be saved for only one Key File, usually the Primary Key File, and always the Key File on which FINDNEW is used to add records. A negative KFAM ID number used as an argument indicates that recovery information is not to be saved for this Key File, and is typically used for all Secondary Key Files. Since records added to the Secondary Key File should be indexed via FINDNEW(HERE) calls which do not update the control information about the END location, the END control sector otherwise written during CLOSE would overwrite valid User File records in the first User File sector. When a negative KFAM ID number argument is present, the END control sector is not written. The Primary Key File is closed with a positive KFAM ID number, which writes the END control sector and recovery information into the User File.

## 7.5 OPEN (DEFFN'230)

The OPEN subroutine is used to open a User File and its companion Key File which were previously created by Initialize KFAM File. OPEN must be executed prior to execution of any other KFAM subroutine. In the OPEN subroutine, a pair of modified ISS Screen/Disk Open subroutines are executed to open the specified User File and Key File, and a SELECT statement is executed to associate each file number with its corresponding disk address. OPEN assigns a specified KFAM ID number to the pair of files. If multiple Key Files are to be used, the primary Key File is opened first, and each Secondary Key File is then opened with a negative Key File number (argument).

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)
GOSUB'230 (I,K,U,F,N$,A,P$, "xxx", "yyy")
```

### For The GOSUB' Statement

"I" is the KFAM ID number (1-8) is to be associated with the newly opened file, and must be used to reference the specified User File and Key File in subsequent KFAM subroutines.

"K" is the file number to be assigned to the Key File, 1 through 9, except for Secondary Key Files where a negative number (-1 through -9) must be specified. The absolute value of the Key File numbers used must be unique.

"U" is the file number to be assigned to the User File.

"F" is the Key File number (the sixth character in the Key File name). If only one Key File is used to index a single User File, it may be an integer from 1 to 9; it is normally 1. If multiple Key Files are to be used, the Primary Key File has a positive Key File number (1-9), and each Secondary Key File is opened with a negative Key File number (-1 through -9). For any User File, the absolute value of F must be unique.

"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 7-1:

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

"P$" is the file password assigned to this file (usually during Initialize KFAM File).

"xxx" is the disk device address, Key File. Hog mode addresses are invalid.

"yyy" is the disk device address, User File (required by KFAM-7 in ISS-5). Hog mode addresses are invalid.

NOTE:

The User File device address is a required argument value. With multiple Key Files, argument values for I, K, U, and F must be unique.

Return Codes for OPEN

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

Q = "X" if the station number (S2) is not 1-16, if the file is already open, if there is a duplicate file number, if the access mode is invalid, if the Key file is not found, or if the disk device address is illegal (hog mode addresses are invalid).

Q = "D" if the named User File is a program file, scratched, or not found.

Q = "M" if the named User File is not a Multiplexed/Multistation file.

Q = "P" if the password is invalid.

Q = "A" if there is an access mode conflict (see Table 7-1).

Q = "S" if the internal global table of open files is full (Single Bank and Multiple Bank versions only). See Chapter 8, Section 8.1, global array @T$(), for additional information.

ERR P34 KFAM ID number not 1-8.

ERR P57 File # not 0-15.

ERR X71 Key File Number not 0-9 or -1 through -9 (0 is invalid but not checked).

## Changing Access Modes

To change a KFAM 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.

## 7.6 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; for blocked records, the variable Q indicates the record within the sector (KFAM variable T6 is set for DA and BA mode access use). 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$)

where: "I" is the KFAM ID number assigned to the file in an OPEN subroutine.

"P" is the record protect option. P = to 1 or 3 indicates record protection; P = to 0 or 2 indicates no record protection. With the Multiplexed KFAM-7 version, P = 2 or 3 also indicates hog request for the Key File's disk drive.

"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. Also, if the Key File hog was requested, the Key File's disk drive is hogged; retry after a 200 ms. wait.

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 increased by one sector. For unblocked files, this can be done as follows:

Suppose the statements:

```
    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:

```
4040 GOTO 4060
4050 $BREAK 5:REM IF BUSY, WAIT BEFORE RETRY (2200MVP)
4060 GOSUB'231 (1, 1, A$): REM DELETE
4065 IF Q$ = "B" THEN 4050: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 the use of special techniques and the FINDNEW(HERE) subroutine. For information on these techniques, see Section 7.9 and Chapter 8, Section 8.5.

## 7.7 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 (KFAM variable T6 is set for DA and BA mode access use). 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$)
```

where: "I" is the KFAM ID number assigned to the file in the OPEN subroutine.

"P" is the record protect option. P = 1 or 3 indicates record protection; P = 0 or 2 indicates no record protection. With the Multiplexed KFAM-7 version, P = 2 or 3 also indicates Key File disk drive hog request.

"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. Also, if the Key File hog was requested, the Key File's disk drive is hogged; retry after a 200 ms. wait.

Q$ = " " ("space") if the key was located successfully.

## 7.8 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 (KFAM variable T6 is set for DA and BA mode access use). For blocked records, variable Q indicates the record within the sector. Refer to Section 7.1 for precedures involved with adding records. Shared or Exclusive access is required. This subroutine adds one record to the record count.

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

where: "I" is the KFAM ID number assigned to the file in an OPEN subroutine.

"P" is the record protect option. P = 1 or 3 indicates record protection; P = 0 or 2 indicates no record protection. With the Multiplexed KFAM-7 version, P = 2 or 3 also indicates Key File disk drive hog request.

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

"0" (zero) is required in the argument list.

### 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 if eight 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. Also, if the Key File hog was requested, the Key File's disk drive is hogged; retry after a 200 ms. wait.

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 (KIRs) are normally split 50/50, where the full KIR is split into two KIRs, each containing a nearly equal number of KIEs. KFAM provides an adjustable "bias" in the form of the user partition KFAM variable V8, which can range from .2 to .8. KFAM sets the bias at .5 when each file is opened. An experienced programmer may set the bias following each Open for subsequent FINDNEW or FINDNEW(HERE) operations; the bias affects all KFAM files accessed by that partition. The variable V8 may be set to any value between .2 and .8. Records entered in ascending key sequence order are packed best at .2, in random order at .5, and in descending key sequence at .8. Also see Chapter 8, Section 8.8.

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"
```

## 7.9 FINDNEW(HERE) (DEFFN'234)

The FINDNEW(HERE) subroutine can be implemented (1) to reuse the User File space occupied by deleted records, (2) to change the value of the key of an existing record, or (3) for adding records to Secondary Key Files. 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 the Delete subroutine unless multiple Key Files are used. To use FINDNEW(HERE) to reuse the User File space occupied by deleted records, see Chapter 8, Section 8.5. 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 Section 7.8).

### 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 if file not opened with Shared or Exclusive access. Also, if the Key File hog was requested, the Key File's disk drive is hogged; retry after a 200 ms. wait.

Q$ = "D" if the specified key 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 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"
```

In the preceding example, a single Key File's key is updated. In the case of multiple Key Files where a record's key is being added to a Secondary Key File, extreme care is required if the KFAM ID number has changed. If the last KFAM subroutine call preceding the FINDNEW(HERE) call was for a different KFAM ID number, immediately before calling FINDNEW(HERE), set STR (T5$(i),1,3)=T4$, where i is the KFAM ID number to be accessed during the FINDNEW(HERE) call. See Chapter 8, Section 8.5 for additional information.

Key File sectors (KIRs) are normally split 50/50, where the full KIR is split into two KIRs, each containing a nearly equal number of KIEs. KFAM provides an adjustable "bias" in the form of the user partition KFAM variable V8, which can range from .2 to .8. KFAM sets variable V8 to .5 when each file is opened. Following each Open, an experienced programmer may set it to any value in the range .2 to .8 for subsequent FINDNEW or FINDNEW(HERE) operations; the bias affects <u>all</u> KFAM files accessed by that partition. Records entered in ascending key sequence order are packed best at .2, in random order at .5, and in descending key sequence at .8. Also see Chapter 8, Section 8.8.

## 7.10 FINDFIRST (DEFFN'235)

The FINDFIRST subroutine sets the User File Current Sector Address to the first record in logical key sequence (KFAM variable T6 is set for DA and BA mode access use). 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)
```

where: "I" is the KFAM ID number assigned to the file in an OPEN subroutine.

"P" is the record protect option. P = 1 or 3 indicates record protection; P = 0 or 2 indicates no record protection. With the Multiplexed KFAM-7 version, P = 2 or 3 also indicates Key File disk drive hog request.

### 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. Also, if the Key File hog was requested, the Key File's disk drive is hogged; retry after a 200 ms. wait.

Q$ = " " (space) if the subroutine executed properly.

## 7.11 FINDLAST (DEFFN'236)

The FINDLAST subroutine sets the User File Current Sector Address to the last record in logical key sequence (KFAM variable T6 is also set for DA and BA mode access use). 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 returns Q$="E"). The calling sequence is:

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

where: "I" is the KFAM ID number assigned to the file in an OPEN subroutine.

"P" is the record protect option. P = 1 or 3 indicates record protection; P = 0 or 2 indicates no record protection. With the Multiplexed KFAM-7 version, P = 2 or 3 also indicates Key File disk drive hog request.

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. Also, if the Key File hog was requested, the Key File's disk drive is hogged; retry after a 200 ms. wait.

Q$ = " " (space) if the subroutine executes normally.

## 7.12 FINDNEXT (DEFFN'237)

The FINDNEXT subroutine sets the User File Current Sector Address to the record immediately following (in logical ascending key sequence) the last record accessed by KFAM (KFAM variable T6 is also set for DA and BA mode access use). For blocked records, the variable Q indicates the record within the sector.

FINDNEXT must be preceded by a FINDNEXT, FINDFIRST, FINDOLD, FINDNEW, FINDNEW(HERE), DELETE, or FINDPREVIOUS subroutine call executed on the specified KFAM file, which did not have a return code of Q$="X" or Q$="E". For instance, when preceded by FINDFIRST, repeated FINDNEXT calls allow ascending key sequence access to the entire file. If FINDNEXT is executed following a FINDOLD call that returned Q$="N" (key not found), FINDNEXT locates the record whose key logically follows the key passed via FINDOLD. If FINDNEXT returns Q$="B" (busy signal), a subsequent FINDNEXT accesses the same (previously busy) record. A DATALOAD statement is executed following FINDNEXT to read the record. The calling statement is:

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

where: "I" is the KFAM ID number assigned to the file in an OPEN subroutine.

"P" is the record protect option. P = 1 or 3 indicates record protection; P = 0 or 2 indicates no record protection. With the Multiplexed KFAM-7 version, P = 2 or 3 also indicates Key File disk drive hog request.

FINDNEXT Return Codes

Q$ = "B" busy signal. The record sought is protected by another station. A retry is recommended. 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. Also, if the Key File hog was requested, the Key File's disk drive is hogged; retry after a 200 ms. wait.

Q$ = "E" if the previous reference was to the last record in logical key sequence (end of file).

Otherwise, Q$=" " (space).

## 7.13 FINDPREVIOUS (DEFFN'212)

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

FINDPREVIOUS must be preceded by a FINDPREVIOUS, FINDLAST, FINDOLD, FINDNEW, FINDNEW(HERE), DELETE, or FINDNEXT subroutine call executed on the specified KFAM file, which did not have a return code of Q$="X" or Q$="E". For instance, when preceded by FINDLAST, repeated FINDPREVIOUS calls allow descending key sequence access to the entire file. If FINDPREVIOUS is executed following a FINDOLD call that returned Q$="N" (key not found), FINDPREVIOUS locates the record whose key logically precedes the key passed via FINDOLD. If FINDPREVIOUS returns Q$="B" (busy signal), a subsequent FINDPREVIOUS accesses the same (previously busy) record.

The calling sequence is:

GOSUB'212 (I,P)

where: "I" is the KFAM ID number assigned in an OPEN subroutine.

"P" is the record protect option. P = 1 or 3 indicates record protection; P = 0 or 2 indicates no record protection. With the Multiplexed KFAM-7 version, P = 2 or 3 also indicates Key File disk drive hog request.

### 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. Also, if the Key File hog was requested, the Key File's disk drive is hogged; retry after a 200 ms. wait.

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.

Q$ = "blank" indicates successful completion.

## 7.14 RELEASE (DEFFN'238)

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

Any call to a KFAM 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 subroutine is called, which automatically releases the protected record.

Where multiple Key Files are used, the RELEASE subroutine must be called once for each KFAM ID number.

The calling sequence is:

GOSUB'238 (I)

where: "I" is the KFAM ID number assigned to the file in an OPEN subroutine.

RELEASE Return Codes

Q$ = "X" if file not open.

Q$ = " " (space) after successful execution.

## 7.15 RE-OPEN (DEFFN'213)

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

The calling sequence is:

GOSUB'213 (I,A,P$)

where: "I" is the KFAM ID 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 (see Table 7-1). 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.

## 7.16 WRITE RECOVERY INFORMATION (DEFFN'214)

WRITE RECOVERY INFORMATION writes the END control sector and the recovery information (most of the KDR) to the next-to-last sector of the User File. Whenever a FINDNEW subroutine is called, the recovery information should be rewritten either automatically when the CLOSE subroutine is called or via WRITE RECOVERY INFORMATION before closing a KFAM file. This operation is normally done when the file is closed if using the KFAM0107, KFAM0207, or KFAM0307 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)

where: "I" is the KFAM ID number.

<u>WRITE RECOVERY INFORMATION Return Codes</u>

Q$ = "X" if file not open.

Q$ = "blank" if executed successfully.

## 7.17 <u>CLOSE (DEFFN'239)</u>

The CLOSE subroutine is used to close a currently open User File and its companion Key File. The KFAM ID number assigned to a closed file can then be reassigned to another file in an OPEN subroutine. Also, the file numbers assigned to a User File and Key File can be reassigned. CLOSE alters all 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 the Close with Recovery Information option was not included when (and if) Build Subroutine Module was used. When using multiple Key Files, the primary Key File should be closed with a positive KFAM ID number, while all Secondary Key Files are closed with a negative KFAM ID number (see Section 7.4). The calling sequence is:

GOSUB' 239 (I)

where: "I" is the KFAM ID number assigned to the file in an OPEN subroutine. Following execution of CLOSE, this number can no longer be used to access the User File and its associated Key File unless the file is reopened. A negative value of argument I indicates the END control sector and recovery information are not to be written.

<u>CLOSE Return Codes</u>

Q$ = "X" if file not open.

Q$ = " " (space) if successful.

NOTE:

CLOSE must be executed promptly at the conclusion of file access operations. If multiple Key Files are used, the file in which FINDNEW is used to add records must be closed with a positive KFAM ID number; this is typically the Primary Key File.

## 7.18 NON-KFAM FILE SUBROUTINES INCLUDED WITH KFAM-7

Please note that the following subroutines are described in Chapter 3 of this manual. These subroutines are available for cataloging KFAM files or for use with non-KFAM data files. Similar access mode capabilities are provided. 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 8
# KFAM-7 TECHNICAL INFORMATION AND ADVANCED PROGRAMMING

## 8.1 KEY FILE AND MEMORY-RESIDENT CONTROL INFORMATION

The first sector of the Key File contains the KDR, which is rewritten when a FINDNEW, FINDNEW(HERE), DELETE, or CLOSE subroutine is executed. The remaining sectors contain as many KIRs, as are necessary to index the User File up to the maximum of eight index levels. An END (end-of-data) control sector and a catalog trailer (end-of-file) control sector follow the last KIR.

### Key Descriptor Record (KDR)

KDR contents occupy the first sector of the Key File, as well as the next-to-last sector (dummy END control sector) of the User File (bytes 3-146, T$2()). These two sectors are identical except for variable V, which is always zero (0) in the KDR on disk, unless the Multiplexed KFAM-7 version is in use. With the Multiplexed KFAM-7 version, the variable V serves as a Key File busy/free flag with the following possible values:

0 = file not busy
1 = busy, FINDNEXT
2 = busy, FINDPREVIOUS
7 = busy, FINDNEW(HERE)
8 = busy, FINDNEW
4 = busy, any other subroutine.

The value of variable V stored in memory (for all versions but the Multiplexed KFAM-7 version), 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 described in Table 8-1.

Table 8-1. The KDR Control Information

| START | LENGTH | $UNPACK VARIABLE | IMAGE | CODE | DESCRIPTION |
|---|---|---|---|---|---|
| 1 | 16 | - | - | - | Indicates completion codes per station (1-16) except: Initialized to "2" by some utilities (KFAM1007, KFAM7007). Following FINDOLD, not found, = "2". Following FINDNEW, FINDNEW(HERE), or DELETE, all values of "9" or less are set to "3". Following a successful Open, set to letter "O". Following successful CLOSE, set to "2". If Q$="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. |

Table 8-1. The KDR Control Information (continued)

| START | LENGTH | $UNPACK VARIABLE | IMAGE | CODE | DESCRIPTION |
|---|---|---|---|---|---|
| 66 | 8 | V1$8 | - | - | Per byte (not used by subroutines):<br>1 = record type is A,B,C,M, or N.<br>2 = record length.<br>3,4 = starting position of key.<br>5 = key length.<br>6 = number of entries in KIR.<br>7,8 = unused |
| 97 | * | T3$3 | - | S | For all versions but the Multiplexed KFAM-7 version: User File, current relative sector for FINDNEW, per station, in first two bytes; also, current record within block for FINDNEW, per station, in last byte. |
| 97 | * | Q4$2 | - | S | For the Multiplexed KFAM-7 version, this indicates the User File current relative sector for FINDNEW, per station. |
| 99 | * | V5$1 | - | S | For the Multiplexed KFAM-7 version, this indicates the User File current record within block for FINDNEW, per station. |

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$ (or Q4$ and V5$) is stored per station. The location is STR(T$(3), 3*S2-2, 3) where S2=station number, in the preceding table.

## Key Index Record (KIR)

Each KIR occupies one sector in the Key File. The number of KIRs in a Key File depends upon the number of Key Index Entries (KIEs), as discussed in Section 8.8. Table 8-2 describes KIR control information.

Table 8-2. KIR Control Information

| 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 KIEs. Number of KIEs per KIR can vary from 7 to 48. Unused KIEs are filled with all bytes HEX(FF). Active KIEs are packed as follows:<br><br>K bytes: key<br>3 bytes: pointer<br><br>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 points to a data record, and is not defined if the pointer is to a lower level KIR. |
| TOTAL | 247 bytes | |

## Internal Storage: User Partition with the Non-Multiplexed KFAM-7 Versions

Certain information stored in the KDR is also stored internally in user partition memory for the last file accessed with all versions but the Multiplexed version.

Variables for KFAM files that are open but not the last file accessed are stored in two arrays, T5$() and V0$() whose array elements in the dimension statement may be set to a value other than 3 (number of KFAM files open). The array element is the KFAM ID number. Variables are packed in field format. V0$(3)21 contains variables set during the OPEN, which remain unchanged (see Table 8-3). @Q8$20 is the hex image for unpacking.

Table 8-3. User Partition Storage of Array V0$() with the Non-Multiplexed KFAM-7 Versions

| VARIABLE | START | IMAGE | DESCRIPTION |
|---|---|---|---|
| V0$2 | 1 | A002 | Absolute starting sector of Key File. |
| V6 | 3 | 5002 | Internal File ID. |
| 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 number, Key File. |
| T2 | 12 | 5002 | File number, 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 contains variables which change with each access, saved every time files are switched (see Table 8-4). @T5$10 is the hex image for unpacking.

Table 8-4. User Partition Storage of Array T5$() with the Non-Multiplexed KFAM-7 Versions

| VARIABLE | START | IMAGE | DESCRIPTION |
|---|---|---|---|
| T4$3 | 1 | A003 | Pointer to record accessed, bytes: 1 - 2 = relative sector address within user file, set equal to variable T6. 3 = record within block, set equal to variable Q. |
| 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 for the Multiplexed KFAM-7 Version

Certain information stored in the KDR is also stored internally in user partition memory for the last file accessed.

Variables for KFAM files that are open but not the last file accessed are stored in two arrays, T5$() and V0$() whose array elements in the dimension statement may be set to a value other than 3 (number of KFAM files open). The array element is the KFAM ID number. Variables are packed in field format. V0$(3)24 contains variables set during the OPEN, which remain unchanged (see Table 8-5). Q8$22 is the hex image for unpacking.

Table 8-5. User Partition Storage of Array V0$() with the Multiplexed KFAM-7 Version

| VARIABLE | START | IMAGE | DESCRIPTION |
|---|---|---|---|
| V0$2 | 1 | A002 | Absolute starting sector of key file. |
| Q0$3 | 3 | A003 | Key File device address. |
| V4$4 | 6 | A004 | Hex image for unpacking entry from KIR, HEX(A0XXA003), where XX = key length. |
| V0 | 10 | 5001 | Access mode (1,2,3,4). |
| T1 | 11 | 5002 | File number, Key File. |
| T2 | 13 | 5002 | File number, User File. |
| T4 | 15 | 5002 | Key length. |
| T5 | 17 | 5002 | KIE length = T4 + 3. |
| V6 | 19 | 5002 | Number of entires in KIR. |
| V7 | 21 | 5002 | KIR bytes used = T5 * V6. |
| V1 | 23 | 5002 | Last key location = V7 - T5 + 1. |

The array T5$(3)58 contains variables which change with each access, saved every time files are switched (see Table 8-6). T5$10 is the hex image for unpacking.

Table 8-6. User Partition Storage of Array T5$()
with the Multiplexed KFAM-7 Version

| VARIABLE | START | IMAGE | DESCRIPTION |
|---|---|---|---|
| T4$3 | 1 | A003 | Pointer to record accessed, bytes: 1 - 2 = relative sector address within User File, set equal to variable T6; 3 = record within block, set equal to variable Q. |
| 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 with the Non-Multiplexed Versions of KFAM-7

All KFAM-7 versions except the Multiplexed KFAM-7 version control access to the disk internally through global tables rather than by reading and writing the KDR (as is done in the Multiplexed KFAM-7 version). This not only saves disk access time but 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, which is called KFAMCOM with the Multiple Bank version and KFAM with the Single Bank version.

In the global area is a table of open files, global array @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 size must be increased if more than 30 KFAM files are to be open. The contents of this table, per entry, are provided in Table 8-7:

Table 8-7. Table of Open Files, Array AT$(), per Entry

| START | LENGTH | CONTENTS |
|---|---|---|
| 1 | 2 | KDR sector (VO$) - starting absolute sector address of Key File. |
| 3 | 1 | One-byte packed disk device address of Key File, packed as described below. |
| 4 | 2 | Catalog trailer (end-of-file) control sector of the User File, where the Multiplexed/Multistation access table is stored. This is an absolute sector address. |
| 6 | 1 | One-byte packed disk device address of the User File, packed as described below. |
| 7 | 1 | Number of index levels, TO, in IBM packed format. |
| 8 | 2 | Relative sector address of highest level index sector, T2$. |
| 10 | 8 | Per station 1 - 16, one half byte for internal completion code. |

Disk device addresses in global array @T$( ) are packed as follows: 1) if the first byte of the address is "3", make the third byte = "1", and 2) the second and third bytes are HEXPACKed, which forms a unique identifier.

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 variable TO (number of index levels), variable T2$ (sector address of highest level index), array T2$() (path through index to current record, in terms of KIR sectors read), and variable T$ (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 TO, T2$ from table @T$(), above. |

| | |
|---|---|
| 8 | Error condition; next and previous records not defined. |
| E | File open. |
| F | File closed. |

The above bit settings are unpacked into variable 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 entry number for a given file within array @T$(). Array @T$(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 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 array @T$() to indicate that the file has been opened. It also stores T0 and T2$ in @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. 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
```

Then 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) for single Key File access to a User File. 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 provided in Table 8-8:

Table 8-8. Table of Protected Records, Array @V4$, per Entry

| START | LENGTH | CONTENTS |
|---|---|---|
| 1 | 1 | Station number, common variable S2, in hexadecimal form. |
| 2 | 1 | Internal KFAM file ID number, variable V6 from array @T$(), the table of open files. |
| 3 | 1 | User file device address, packed. |
| 4 | 2 | User File sector identifier, obtained by taking the catalog trailer control sector absolute sector address minus (SUBC) the relative sector address within the User File. |

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; normally, 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 should therefore not slow down performance very much.

## 8.2 KFAM-7 FILE NAMES

File names for the KFAM-7 diskette are provided in Table 8-9. All files are program files (modules) unless indicated otherwise.

Table 8-9. Description of the Files on the KFAM-7 Diskette

| DESCRIPTION | FILE NAME |
|---|---|
| Work file used to generate code and store messages to be printed by KFAM2107. This is a data file. | KFAMWORK |
| ISS start-up station file, used to create other station files. This is a data file. | ISS.000D |
| ISS start-up station file, contains utilities default values for this station number (nn). This is a data file. | ISS.0nnD |
| KFAM variables and subroutines, Single Bank version. | KFAM0107* |
| KFAM variables and subroutines, Multiplexed version. | KFAM0207* |

Table 8-9. Description of the Files on the KFAM-7 Diskette (continued)

| DESCRIPTION | FILE NAME |
|---|---|
| KFAM subroutines, Multiple Bank version. | KFAM0307* |
| KFAM variables, Multiple Bank version. | KFAM0407** |
| ISS start-up, application loading menu. | ISS.000A |
| ISS start-up, station number. | ISS.000M |
| ISS start-up, station parameters. | ISS.001M |
| ISS start-up, system menu. | ISS.002M |
| ISS Data Entry subroutine. | ISS.200S |
| ISS Date subroutine. | ISS.220S |
| ISS Operator Wait subroutine. | ISS.254S |
| ISS start-up load module. | START |
| (Other file names are the KFAM Utilities) | |
| * Indicates self-designated "KFAM" global partition, a global partition. **Indicates self-designated "KFAMCOM" global partition, a universal global partition. | |

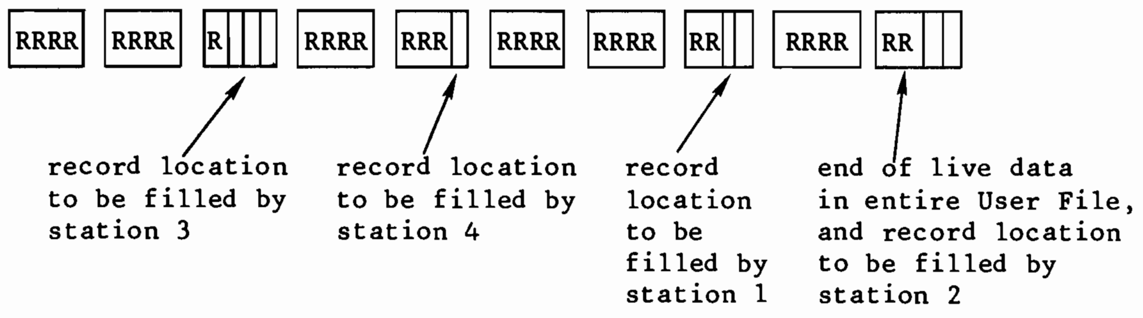
## 8.3 FINDNEW WITH BLOCKED FILES

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.

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. The 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, 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 FINDNEWs 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, the User File might look like this:


RRRR
RRRR
R
RRRR
RRR
RRRR
RRRR
RR
RRRR
RR
record location to be filled by station 3
record location to be filled by station 4
record location to be filled by station 1
end of live data in entire User File, and record location to be filled by station 2

where: R = record
□ = unoccupied record locations

## 8.4 FILES TOO LARGE FOR ONE DISK

A cataloged disk file must be entirely 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. One Key File must be created 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 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 and using a variable for the KFAM ID number in argument lists for record access subroutine calls.

For example, an inventory file is split into two files; all keys lower than 400962-B4 reside in the first file at disk address 320, while higher keys reside in the second file at address B20. The variable X is used to denote the KFAM ID number (argument) for a FINDOLD subroutine call, and the value X+Z identifies the corresponding User File (UF) number, as selected at line 500. Note the presence of the DEFFN'31 statement followed by a RETURN CLEAR statement (lines 760-780) to allow SF'31 to be touched to indicate completion. Error handling routines (not shown) could be provided beginning at line 800, which might display the subroutine's name and the error code (Q$) associated with the error, and then close the files.

## 8.5 REUSING DELETED SPACE WITH FINDNEW(HERE)

Immediately following a DELETE, a FINDNEW(HERE) call 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 8.1).

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 ID number. If not sure which file was last accessed, test T9 = KFAM ID number last accessed.

## 8.6 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 8.1.)

There are legitimate reasons why a user may wish to change information in the KDR. One problem which could occur is that the starting position of the key or the record length is wrong, causing a reorganization program to fail. These fields are critical in reorganizing the KFAM file and 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 these 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.

## 8.7 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 control information originally written during the Initialize KFAM File utility. Two control bytes in this control sector mark it as an END control sector 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, KFAM0207, or KFAM0307 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 8.1 for the KDR. |

## 8.8 KEY INDEX RECORDS AND THE ADJUSTABLE BIAS

The Key File structure 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 KIEs 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 KIEs. As new keys are added to the file, their KIEs 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. For this example, it has been assumed 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 added key, 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" must be introduced. Both KIRs so far created are on Level 1, the lowest level. The lowest level is defined as the level containing 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 each KIR sequentially for a given key, the system builds a tree structure to minimize search time. There is one and only one KIR at the highest level, whose sector address is recorded in the KDR. The search is started by reading this sector. Prior to the creation of the second KIR, the search was completed by locating the position of the key within the one sector. But at this point, there are two KIRs 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 KIRs, HEX(0001) and HEX(0002). This level-2 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. Variable T0 is now = 2, to show that the index now has 2 levels. Variable 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. The key 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; and it therefore falls in the range 000 to 670. Since 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 the value of variable 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 KIRs on Level 1 will again become full, and the KIR must again 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 cause Sector 1 to become full, and the addition of 407 makes 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 a new level is not yet necessary. In this example, there is room in the Level-2 index to reference up to 15 level-1 KIEs. 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 Level-2 index is full, it is split in the same way the original KIR was split, and a third level is created, pointing to two Level-2 KIRs, which in turn point to the Level-1 KIRs. The Level-1 KIRs 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.

Assuming a Key File has an average of 10 KIEs per KIR, the average number of records accessed by a given number of index levels 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 seven KIEs and a guaranteed average minimum of four KIEs; the maximum eight levels of index access at least 65,535 records for a "worst case" 30-byte key.

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, 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 KIEs 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 KIEs are divided between the old and new KIRs. 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 KIEs. 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, its KIEs are always divided equally unless a programmer sets the adjustable bias. Consider keys which are being added sequentially. Again assume the first index sector is filled by 14 active KIEs 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 half-full KIRs (50% 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 KIRs each containing 12 KIEs or approximately 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.

However, 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 KIRs already containing 12 entries because of the greater range of values represented. Consequently, 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 KIEs 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 KIEs per KIR to give the split; i.e., the number of KIEs which go into the new KIR. The adjustable bias may range from .2 to .8 and is set at .5 following the Open by KFAM.

Following each KFAM OPEN, the programmer may set the adjustable bias, user partition variable V8, to a value from .2 to .8. However, remember that KFAM resets V8 following each OPEN to .5 and be aware that V8 is used as the adjustable bias for all KFAM files being accessed by that partition. In general, consider resetting V8 before each series of FINDNEW or FINDNEW(HERE) subroutines are to be performed.

On the basis of past experience, the bias should approach .2 as keys are added in ascending key sequence, should stay at .5 if keys are added in random order, and should be at .8 if records are added in descending 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 Build Key File, the bias is initially set to .5 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: keys are duplicated in higher level indexes, and a bias is available for the splitting of KIRs.

## 8.9 PROCEDURES FOR CHANGING USER FILE RECORD LAYOUT

When a KFAM file's data record layout is to be changed, (to accommodate the addition of a new field, for instance) it is generally recommended that a new KFAM file be created to accommodate the new record format. Such a change requires modification of the argument list used for DATASAVE and DATALOAD statements for the User File, and usually affects the critical system information defined during Initialize KFAM file and stored in the Key File's KDR.

The procedure requires that the new KFAM file be initialized using Initialize KFAM File to define the new logical record length, file type, key location, etc., and create (catalog) the new User File and Key File. A simple user-supplied program can be written to read the records from the old User File using the old DATALOAD argument list, and save the records into the new User File using the new DATASAVE statement argument list by using the following procedure:

1. Open the two KFAM files by calling the OPEN subroutine once for each file. Set the adjustable bias, variable V8, to .2.

2. Call the FINDFIRST subroutine for the old KFAM file to locate its first record.

3. Execute a DATALOAD statement for the old User File using the old argument list to read the first record.

4. Update the values of the variables to be used in the new DATASAVE statement argument list, if necessary.

5. Call the FINDNEW subroutine for the new file, which adds its key to the new Key File and returns the position where it is to be written.

6. Execute a DATASAVE statement for the new User File using the new argument list to write this record.

7. Call the FINDNEXT subroutine for the old KFAM file to locate its new record in logical key sequence.

8. Check if Q$="E" upon return. If Q$="E" skip to Step 10. Otherwise, continue with Step 9.

9. Execute a DATALOAD statement for the old Key File to read the next record. Go to Step 4.

10. When Q$="E" after a FINDNEXT subroutine call, this indicates the record previously read had the highest key value. Both files should now be closed by calling the CLOSE subroutine once for each file.

The new KFAM file should be checked by running the Print Key File utility. The new KFAM file might be copied using the ISS Copy/Verify utility to the disk address of the old KFAM file, and replace the old KFAM file using the output mode of replace, followed by the Reallocate File Space utility.

Applications software written for the old file would require changes to the DATALOAD and DATASAVE statements used to access User File records and possibly to the DIM and COM statements used to dimension argument list variables. The LIST T form of the LIST command is helpful in locating the live numbers containing these statements; for instance, entering: LIST T "DATA" lists all line numbers containing the four letters DATA (as found in DATALOAD and DATASAVE statements) for the program text currently in memory. Each line might then be recalled and modified if required, before resaving the program text to disk.

# APPENDIX A - KFAM UTILITY ERROR MESSAGES AND RECOVERY PROCEDURES

KFAM Utilities provide many error messages, each of which indicate a different condition. In general, these error messages fall into one of three error categories, as listed 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 (such as an ERR 1nn type error), is rarely encountered, and requires the operator to touch SF'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 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 immediately appears 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, run the Reset Access Tables utility.

Table A-1. KFAM 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.<br><br>A-2. To escape from (abort) this program, touch SF'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 | B. Touch SF'31 once to close all files. Refer to 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 SF'31 to bring the KFAM menu to the screen.<br><br>C-2. Otherwise, the operator may touch CLEAR and RETURN and then load a program. |

NOTE:

If the error recovery procedures require SF'31 to be touched, Edit mode (blinking cursor) must be switched off (steady cursor) before SF'31 is touched. Edit mode is manually switchable by means of the EDIT key.

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.

Table A-2. KFAM Utility Error Messages

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| ACCESS ERROR<br><br>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.<br><br>Could also 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. |
| ADDRESS '   ' HAS NOT BEEN INCLUDED IN ISS START-UP | Reset Access Tables requires and ISS start up printer address other than blank. | Touch SF'31 repeatedly until the start-up display appears. Change the printer address and retry. |
| 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 Chapter 5, Section 5.5 for further explanation; applies to record types A, B, C. | Recalculate record length, blocking factor, if required. Re-enter the logical record length and continue. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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 RETURN to resume. The 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, as denoted by different error codes (1nn format). Should a statement line beginning with an "at" sign (@) appear, the error was encountered during global program text execution and may be caused by an application program error or an erroneous KDR. | See Table A-1 and the errors listed below for certain conditions; refer to the Wang BASIC-2 Language Reference Manual for other errors. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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/Rebuild Sub-System where #1 indicates User File and #2 indicates Key File). For all utilities, #0 indicates sector in 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 and humidity). | Note the file number, then touch SF'31 to close file. See Table A-1. In general, restore same disk or create new disk from backup copy. Rerun the utility.<br><br>With Reorganize/ Rebuild Subsystem, run Key File recovery oninput User File. If Part 3, reorganize output User File, assigning its name as the input User File name, and then rerun Reorganize/Rebuild Subsystem. If error persists, file is permanently damaged, or hardware malfunc-tion 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). For all utilities, #0 indicates sector on KFAM system disk; #T1 or #T1(T9) indicates Key File sector. This | Record the file number, and depress SF'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/Rebuild Subsystem, Part 1: Output User File contains a bad physical sector. Part 2: Output Key File contains a bad |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| | indicates which disk may be defective. | physical sector. Part 3: Input User File contains a bad physical sector. For Key File 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-5/KFAM-7 file was accessed, e.g., KFAM-3, KFAM-4 file. | Touch SF'31 to close files, then see Table A-1. Either use the Convert To KFAM-7 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. |
| ERR X77 | Attempted to select a global subroutine which has not been loaded and run. | Touch SF'31. Load and run the appropriate global program file before reattempting this operation. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| ERROR#XX LINE XXXX (also ERR XX) | The error code and line number are displayed. Refer to typical ERR lnn 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 the user file name, or 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 setup module) to "C" and rerun. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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 Tables is required. Run Reset Access Tables if the file was 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 the user file name and continue. If Key File not found, re-enter the key file number and continue. If this error persists, check if correct disk is at the specified disk address. |
| FILE NOT FOUND<br>USE ISS FILE STATUS<br>REPORT TO CLOSE FILES | The utility could not locate the catalog trailor control sector of one of the files. | Touch SF'31. Use the File Status Report ISS utility to close all files which are open to this station number. |
| FILE XXXXXXXX NOT FOUND<br>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. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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. |
| GLOBAL 'KFAM' SUBROUTINES NOT AVAILABLE - OVERLAYS WILL BE USED | A global "KFAM" partition is not running. If overlays are to be used, sufficient memory is required. | To proceed using overlays, continue. To load and run the appropriate global "KFAM" program file, touch SF'31 to obtain the system menu. From the appropriate partition, clear, load, and run the global file. Then retry. |
| INPUT AND OUTPUT USER FILE MAY NOT BE THE SAME FILE | Both input and out-put 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 setup 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 Tables; then rerun this program. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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 the 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 setup module (program). Rerun this program. |
| INVALID KEY<br><br>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 the Description column) by completing the recovery procedures described for SEQUENCE ERROR (Reorganize In Place)(e.g., write a program to compare keys, or to correct record length, key position in KDR). |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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. Other-wise, touch SF'31 to abort this program. Check for errors in application programs that may have caused this con-dition. Run Key File Recovery on input User File. Rerun this program. If unsuccessful, see INVALID KEY. 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 the 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. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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 dummy END control sector 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 Build Key File 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/ Rebuild Subsystem, the applications programmer should |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| | | re-create the file if the User File is wrong or re-create file parameters in the KDR (see Chapter 8, Section 8.6). 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 setup 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. If this is a duplicate key and the key length and starting position of the key were carefully calculated, touch RETURN, touch SF'1 to change the key type to Duplicate, and then retry. Otherwise, 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 NOT INITIALIZED | The specified Key File has not been initialized. | See Table A-1. Run Initialize KFAM File. Then reun this utility. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| KEY FILE SPACE EXCEEDED<br><br>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 either (1) increasing extra sectors with Copy/Verify and then run Reallocate File Space, or (2) run Reorganize/Rebuild Subsystem with variables 04$ and 04 set to create a larger output Key File. 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. If this is a duplicate key and the starting position of the key and the key length were calculated carefully, touch RETURN, touch SF'1 to change the key type to Duplicate, and then retry. Otherwise, re-enter the starting position of the key or 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 key length or starting position of the key. If this is a duplicate key and the starting position of the key and the key length were calculated carefully, |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| | | touch RETURN, touch SF'1 to change the key type to Duplicate, and then retry. Otherwise, re-enter the starting position of the 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; if the file was accidentally left open, run KFAM-4 version of Reset Access Tables. Rerun this program. |
| LAST KEY NOT FOUND<br><br>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.<br><br>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 |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| | | non-matching keys should be corrected or deleted. Then rerun this program. |
| LAST KEY NOT FOUND | Build Key File: 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 Build Key File 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 SF'31, then see Table A-1. Use Reorganize/Rebuild 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 Build Key File, 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 File Space. Rerun this program. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| NO SPACE ON DISK FOR KEY FILE | There is insufficient space on this disk to catalog the Key File. | See Table A-1. With Build Key File, 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 Build Key File 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/ Rebuild Subsystem, the application programmer should re-create the User File if it is wrong, or re-create file parameters in the KDR (see Chapter 8, Section 8.6). 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. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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. |
| OPERATOR INTERRUPT | Program was interrupted by the operator depressing SF'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 Tables, and then rerun this program. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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 success-fully). | 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. |
| OUTPUT USER FILE SPACE EXCEEDED | Space allocated for output User File in setup module is too small. | See Table A-1. Correct setup module program. Either let Reorganize/Rebuild 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. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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/Rebuild 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. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| SEQUENCE ERROR<br><br>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 disk. |
| SYSTEM ERROR | Hardware or software error. | Touch SF'31 and see Table A-1. Rerun this program. Notify Wang Laboratories, Inc., if this error persists. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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 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 the estimated number of records with a smaller number and continue. After completion of this program, to increase the allocation for this User File, use either Reorganize/ Rebuild Subsystem with variables 03$ and 03 set to create a larger output User File, or Copy/Verify followed by Reallocate 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 automatic- ally begins and continues without operator inter- vention. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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 SF'31. Then run ISS Copy/Verify with enough extra sectors to increase allocation of Key File; then run Reallocate 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; if the file was accidentally left open, either use Reset Access Tables, File Status Report, or execute a non-KFAM file Close subroutine to close KFAMWORK. Rerun this program. |

Table A-2. KFAM Utility Error Messages (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| 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

| N \ M | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| : | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |  | X |  | X | X | X |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X |  | X |  |  |  | X |  | X |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X |  | X | X |  |  |  |  | X |  |  |
| 3 | INPUT |  |  |  |  |  | OUTPUT |  |  |  |  |  | X | X |  | X |  | X |  | X |  |  |  | X |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X | X | X |  | X |  |  |  | X |  |  |
| 5 | FILE |  |  |  |  |  | FILE |  |  |  |  |  | X | X |  | X |  | X |  | X |  |  |  | X |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X | X | X |  |  |  | X |  | X |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X | X | X |  |  |  | X |  | X |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X | X | X | X |  |  |  |  | X |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X | X | X | X |  |  |  |  | X |  |  |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X |  |  |  | X |  | X |  | X |  |  |

NUMERIC ARRAYS
FORMAT = MN(

| N \ M | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| : |  | X | X |  |  | X |  |  | X | X | X |  |  | X | X |  |  |  |  |  |  |  |  |  |  | X |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | INPUT |  |  |  |  |  | OUTPUT |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | FILE |  |  |  |  |  | FILE |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

ALPHA NUMERIC SCALARS
FORMAT = MN$

| N \ M | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | X | X | X | X | X | X | X | X | X |  | X | X | X | X | X | X | X |  |  | X |  |  |  | X | X | X |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X |  | X |  | X |  | X |  | X | X |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X | X | X |  | X |  | X |  | X |  |  |
| 3 | INPUT |  |  |  |  |  | OUTPUT |  |  |  |  |  | X | X |  | X | X |  |  |  |  | X |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X |  | X |  | X |  | X |  |  |  |  |
| 5 | FILE |  |  |  |  |  | FILE |  |  |  |  |  | X | X |  | X | X |  |  |  |  |  |  | X |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  | X |  | X | X | X |  |  |  |  | X |  | X |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X |  |  |  |  | X |  |  |  | X |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X |  |  |  |  |  | X |  | X |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X |  |  | X |  | X |  | X |  | X |  |  |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X | X |  |  |  |  | X |  | X |  |  |

ALPHA NUMERIC ARRAYS
FORMAT = MN$(

| N \ M | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| : | X | X |  | X |  | X |  |  |  | X |  |  |  | X | X | X |  |  |  | X |  | X |  | X |  | X |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X |  |  |  |  |  |  |  | X |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X |  |  |  | X |  | X |  | X |  |  |
| 3 | INPUT |  |  |  |  |  | OUTPUT |  |  |  |  |  | X | X | X | X |  |  |  |  |  |  |  | X |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X |  |  |  |  |  | X |  | X |  |  |
| 5 | FILE |  |  |  |  |  | FILE |  |  |  |  |  | X | X | X | X |  |  |  |  |  | X |  | X |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  | X |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  | X |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  | X |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X |  |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X |  |  |  | X |  |  |  | X |  |  |

# 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.

## 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_1$, $X_2$) 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 array variables appearing 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. |

## APPENDIX D - TRANSLATION TABLES: ASCII TO EDCBIC AND EBCDIC TO ASCII

The Translation Table subroutines facilitate character code conversion between ASCII and EBCDIC (see Chapter 3). Supported character conversions appear in Table D-1.

Table D-1. ASCII/EBCDIC Translation

| CATEGORY | DESCRIPTION |
|---|---|
| Alphanumeric Characters | All uppercase letters (A-Z), lowercase letters (a-z), and digits (0-9) are translated. |
| Punctuation Symbols | The following punctuation characters are successfully translated: period (.), question mark (?), comma (,), colon (:), semicolon (;), double quotation mark ("), and single quotation mark or apostrophe ('). |
| Special Characters* | The following special graphic characters are successfully translated: pound or number sign (#), dollar sign ($), percent sign (%), ampersand (&), asterisk (*), slash or division sign (/), at sign (@), plus sign (+), hyphen or minus sign (-), equal sign (=), left parenthesis ((), right parenthesis ()), underscore (_), less than sign (<), greater than sign (>), left brace ([), right brace (]), back slash (\\), grave accent ( ' ), tidle (~), and split vertical line (¦). |
| Control Characters | The following control characters are successfully translated: ACK, NUL, SO, SOH, SUB, BS, SI, STX, ESC, HT, DC1, EOT, DEL, CR, DC2, NAK, BEL, LF, DC3, EM, VT, DC4, FS(IFS), FF, GS(IGS), CAN, SUB, RS(IRS), US(IUS), ETB, ETX, ENQ, SYN, and DLE. |

Table D-1. ASCII/EBCDIC Translation (continued)

* The up arrow, left bracket, and right bracket in the ASCII set and have no EBCDIC counterparts, whereas the logical NOT sign (if defined), cents sign, and logical OR sign in the EBCDIC set have no ASCII counterparts. Therefore, during ASCII to EBCDIC translation, an ASCII up arrow (↑) is translated to EBCDIC hexadecimal 5F (sometimes defined as a logical NOT sign (¬)), a left bracket ([ ) is translated to hexadecimal AD, and a right bracket (]) is translated to hexadecimal BD. Conversely, during EBCDIC to ASCII translation, an EBCDIC hexadecimal 5F is translated to an ASCII up arrow (↑), a cent sign (¢) is translated to a hexadecimal FF, and a logical OR sign (|) is translated to a hexadecimal FF.

## APPENDIX E - ISS UTILITIES ERROR MESSAGES AND RECOVERY PROCEDURES

ISS utilities error messages are listed in Table E-1. If an error message includes a file name, "filename" appears in the listed error message indicating where the actual file name would appear. Similarly, error messages which include a line number are listed with "nnnn" to indicate where the line number would appear. Error messages are listed in alphabetical order.

Table E-1. ISS Utilities Error Messages and Recovery Procedures

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| ALL SPECIFIED PROGRAMS PROCESSED | The specified pairs of program files were identical. | None. This is an informative message. |
| CRT ALLOWS VERTICAL DUMP ONLY. KEY RETURN(EXEC) TO RESUME? | With displayed (CRT), output, only the Vertical dump is allowed. | Touch the RETURN key. Change the Dump Option to vertical and retry or change the output device to printer and retry. |
| CURRENT MEMORY SIZE TOO SMALL BY nnn ENTRIES. | The requested number of entries exceeds the capabilities of this station's memory size by nnn entries. | Retry with a reduced number of entries or, if using a 2200MVP, retry later using a larger partition size. |
| DATA STRUCTURE FOR DATA FILES ONLY. | Program files cannot be dumped using the Data File Structure dump option or if the input mode is Range. | Re-enter the correct file name. |
| ERR A01 | Insufficient memory available to perform the specified operation. | If a 2200MVP is in use, refer to Table 1-1 for the partition size necessary to execute the operation chosen. Touch SF'31. If possible, use a station with a greater memory ize and retry. Otherwise, if using a 2200MVP, either execute a different partition configuration after all stations have completed their operations or wait |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| | | until the next time a partition configuration is to be executed and increase the partition size allocated to this station, then retry. |
| ERR D82 | One of the ISS program files is not contained on the ISS disk(ette) in use. This error should not occur if ISS was copied properly to the disk(ette) in use, if the ISS loading address is correct, and if the correct disk address was selected via a SELECT DISK statement prior to loading ISS software. | Note the file name being pointed to by the error message. Touch SF'31, and reload ISS software; if unsuccessful, make a backup copy of ISS software from the issued diskette to a different disk(ette) following the procedures in Chapter 1 and use the new ISS disk(ette). If this error persists, contact Wang Laboratories. |
| ERR D85 | Either the output disk(ette) accessed has a disk catalog index, or a disk(ette) accessed has not been scratched since it was last formatted. | Touch SF'31. In most cases, retry using a different disk(ette). Later, in the Immediate mode, use the LIST DC statement to list the catalog of the disk(ette) causing this error. If INDEX SECTORS = 00000 appears, this indicates that the disk has not been scratched (use SCRATCH DISK statement). If the disk(ette) has a full catalog index, scratch unwanted files, and use the Copy/Verify utility to copy all files (MODE=ALL) to an output disk with a greater index sectors value. |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| ERR I93 | The disk(ette) accessed was either not formatted, was incorrectly formatted, or has a format error. | Touch SF'31. If using a diskette, remove and gently remount the diskette; retry the attempted operation. Otherwise, retry using a different disk(ette). As a last resort, reformat the disk(ette) causing this error. |
| ERR I95 | A tab was not in place over the write protect hole while attempting to write on the specified diskette. This error can occur during ISS start-up or utility operation when defaults are saved if the diskette is write protected (tab not in place). This error could also be caused by a disk hardware (seek) problem. | Touch SF'31. Remove the diskette and place a tab on the diskette; remount the diskette and retry the attempted operation. If the tab was in place, carefully remount the diskette and retry. If unsuccessful, retry using a different disk(ette). Should this error persist when using the same drive, contact a Wang Service Representative. |
| ERR I98 | A disk(ette) was accessed which was not mounted or, if a diskette drive was specified, the drive door was not closed. Also could be caused by using a diskette in a Model 2270 Diskette Drive where the diskette had been scratched (SCRATCH DISK statement) using a Model 2270A (or 2270AD) Diskette Drive. | Touch SF'31. Check if the diskette is mounted. Retry taking care when specifying the disk address(es). If using a diskette in a Model 2270 Diskette Drive and it may be possible that the diskette was formatted on a different diskette drive, run the Sort Disk Catalog utility; if the END CAT. AREA is greater than 1023, the operation requested can only be performed on a Model 2270A (or 2270AD). |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| ERR P48 | A peripheral device address was specified which does not exist in this 2200MVP's Master Device Table. | Touch SF'31. Do not use the specified address until it has been entered into the Master Device Table; i.e., during partition generation. |
| ERROR IN OPEN | Either the INPUT ONE or INPUT TWO file is (1) a data file, (2) does not exist at its respective disk device address, or (3) exists as a scratched file at its respective device address. The file names specified for INPUT ONE and INPUT TWO and their respective disk device addresses precede this error message. Only program files may be compared. | Were the correct file names specified? Was the correct disk accessed? Run the Sort Disk Catalog utility for both disk addresses to determine the status of the files causing the error message. Retry with correct file names or disks if appropriate. |
| ERROR IN REFERENCE FILE OPEN = A KEY RETURN(EXEC) TO RESUME? | The reference file whose name was entered is currently being accessed by another user or the file was left open due to an operational accident. | Check if another station is accessing the specified reference file. Either wait until that station is done and touch RETURN or touch SF'31 and perform other processing (retry later). If no station is accessing the specified reference file, touch SF'31 and use File Status Report to close the file accidentally left open. |
| ERROR IN REFERENCE FILE OPEN = M KEY RETURN (EXEC) TO RESUME? | The file whose name was entered is not a reference file and not a multiplex/multistation file. | Either mount the correct disk, touch the RETURN key, or touch SF'31 and retry entering the correct reference file name. |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| ERROR IN REFERENCE FILE OPEN = P KEY RETURN(EXEC) TO RESUME? | The reference file whose name was entered is not an ISS-4 reference file; it is an Old (pre-ISS-4) reference file. | Either mount the correct disk and touch the RETURN key or touch SF'15 and retry with correct disk address or reference file name. If unsuccessful, load the Create Reference File utility and choose the edit option. Specify the same file name and the type as old/new. Touch SF'16. When the Utilities menu appears, retry the previous operation. |
| FILE filename ALREADY EXISTS ON PLATTER | When creating a new reference file, the specified file name (which appears in the error message) already exists at the specified disk address. | Retry either choosing the edit option instead of the create option (touch SF'15) or specify a different file name. If the correct disk was not accessed, mount the correct disk and then retry. |
| FILE-filename CANNOT BE COMPRESSED - ERROR IN OPEN | There are two several possible reasons why the filename appearing in the error message could not be compressed: | |
| | With MODE = INDIRECT, the file whose input file name appears in the error message may not be an active program file on the input disk. The file was not compressed. | Check if the correct input disk was accessed. Run the Sort Disk Catalog utility to list the input disk's catalog. Run the Create Reference File utility to print the contents of the specified reference file. Was the correct reference file name specified? Is a reference file of the same name contained on a |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| | | different disk? If wrong disk or reference file name, retry using correct disk or reference file name. If the program file has been scratched, check whether a new file name was assigned to that program. |
| | The program file whose output file name appears in the error message could not be compressed possibly because of one of the following reasons related to the output disk: (1) insufficient disk space on the output disk, or (2) the output file name already exists on the output disk. | Check if the correct output disk was accessed. Run the Sort Disk Catalog utility to list the output disk's catalog. Compare the sector numbers for END CAT. AREA and CURRENT END; if their difference is less than the number of sectors required by the file, retry using a different output disk. If the specified output file name already exists on the output disk, retry using a different output file name. |
| FILE-filename CANNOT BE COMPRESSED - FILE LENGTH ERROR | This is a program error caused by numerous statements in the input program file (filename) which cannot be appended onto other lines in the program text. This causes the output file allocation estimate made by the Compression utility to be too small to copy the compressed program text. | It is recommended that blank REM statements be eliminated from the input program file and the program file be resaved to disk. Retry the compression. If unsuccessful, the user may modify ISS module ISS.031U, statement line 550. The statement E1 = MAX(1.1*A3,A3+5) should be altered to increase the value of E1, e.g., E1 = MAX(A3*1.2, A3+15) and the program file ISS.031U resaved in |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| | | place of the original file; e.g., SCRATCH T/xyy, "ISS.031U":SAVE DC T/xyy, () "ISS.031U". |
| FILE-filename CANNOT BE COMPRESSED - LINE LENGTH ERROR (nnn) | A statement line exceeding 180 bytes was encountered in the input file when the maximum compressed line length was specified as 180. The byte length of the statement line which exceeds the 180 byte limit appears in parentheses; e.g., (nnn). | Either specify the maximum line length as 256 or decompress this program file before compressing it (or both). |
| FILE-filename-CANNOT BE COPIED | There are four possible reasons why the file name appearing in the error message could not be copied:<br><br>With OUTPUT OPTION = ADD, the cause may be insufficient disk space on the output disk to copy the file whose input file name appears in the error message. | Check if the correct output disk was accessed. Run the Sort Disk Catalog utility to list the catalog of the output disk. Compare the numbers for END CAT. AREA and CURRENT END; if their difference is less than the number of sectors required for the file, retry using a different output disk or specify MODE = PART and specify fewer extra sectors. |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| | With OUTPUT OPTION = ADD, the specified output file name may already exist on the output disk. The file whose input file name appears in the error message was not copied. | Check if the correct output disk was accessed. Use Sort Disk Catalog to list the output disk's catalog. If the output file name exists on the output disk, retry retry using either a different output file name or a different output disk. Also, if the file on the output disk may be replaced (overwritten) by the input file, specify the output option as replace or add/replace and retry using MODE = PART. |
| | With OUTPUT OPTION = REPLACE, the specified output file name may not exist on the output disk. The file whose input file name appears in the error message was not copied. | Check if the correct output disk was accessed. Run the Sort Disk Catalog utility to list the catalog of the output disk. If the output file name does not exist on the output disk, retry using a different output disk or output file name; or, specify output option = add or add/replace with the same parameters and retry. Specify MODE=PART. |
| | With OUTPUT OPTION = REPLACE, the specified output file may be too small to contain the contents of the input file and the requested number of extra sectors. The file whose input file name appears in | Run the Sort Disk catalog utility to list the catalog of the output disk. If the output file name exists on the output disk, either retry with a reduced number of extra sectors or retry |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| | the error message was not copied. | specifying output option = add or add/replace and a different output file name. Specify MODE = PART. |
| FILE-filename-CANNOT BE DECOMPRESSED - ERROR IN OPEN | With MODE = INDIRECT, the file whose output file name appears in the error message may not be an active program file on the input disk. The file was not decompressed. Other possible reasons appear below: | Check if the correct Input disk was accessed. Run the Sort Disk catalog utility to list the output disk's catalog. Run the Create Reference File utility to print the contents of the specified reference file. Was the correct reference file name specified? Is a reference file of the same name contained on a different disk? If wrong disk or reference file name, retry using correct disk or reference file name. If the program file has been scratched, check whether a new file name was assigned to that program. |
| | The program file whose output file name appears in the error message could not be decompressed possibly because of one the following reasons related to the output disk: (1) insufficient disk space on the output disk, or (2) the output file name already exists on the output disk. | Check if the correct input disk was accessed. Run the Sort Disk Catalog utility to list the output disk's catalog. Compare the sector numbers for END CAT. AREA and CURRENT END; if their difference is less than the number of sectors required for the, retry using a different output disk. If the specified output file name already exists |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| | | on the output disk, retry using a different output file name. If the output disk was not on-line, retry using the same parameters. |
| FILE-filename CANNOT BE DECOMPRESSED - FILE LENGTH ERROR | Due to the presence of numerous multistatement lines in the input program file, the output file could not be decompressed using the extra sectors value specified. | If sufficient space exists on the output disk, increase the extra sectors value and retry. As an alternative to greatly increasing the required number of extra sectors, after the Utilities menu appears, clear the station in use, load the specified program file, renumber the program text, break up some of the multi-statement lines, save the modified program to disk, and then decompress the program file. |
| FILE - filename - DIFFERENT NUMBER OF SECTORS USED | Either the input and/or output file does not have an END control sector written by a DATASAVE DC END or subroutine equivalent, or the number of used sectors (as determined by the position of the END control sector) is not the same in the input file and output file. The pair of files indicated by the input file name which appears in the error message do not verify. | Check if the correct disk was accessed. Retry taking care when entering the input and output file names to be verified. If un-successful, Copy/Verify the file. |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| FILE - filename - DOES NOT EXIST ON INPUT PLATTER | The specified input file name does not exist on the specified input disk. The file whose input file name appears in the error message could not be copied or verified. | Check if the correct input disk was accessed. Run the Sort Disk Catalog utility to list the catalog of the input disk. Run the Create Reference File utility to print the contents of the specified reference file. Was the correct reference file name specified? Is a ref- erence file of the same name contained on a different disk? If wrong disk, retry using different input disk. Otherwise, retry with the correct reference file name. |
| FILE - filename - DOES NOT EXIST ON OUTPUT PLATTER | During the attempted Verify operation, the output file name specified does not exist on the output disk. The file whose output file name appears in the error message could not be verified. | Check if the correct output disk was accessed. Run the Sort Disk Catalog utility to list the output disk's catalog. If the output file name does not exist on the output disk, retry using either a different output file name or output disk. |
| FILE - filename - DOES NOT VERIFY | The data contained within the pair of files indicated by the input file name appearing in the error message does not verify. The pair of files does not verify. | Check if the correct disk was accessed. Retry taking care when entering the input and and output names to be verified. |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| FILE filename IS NOT AN ACTIVE DATA FILE | When editing or printing (edit or print option) a reference file, the specified reference file name (appearing in the error message) is not an active data file located at the specified disk address, i.e., it is not cataloged, scratched, or is a program file. | Check if the correct disk accessed. Change the specified file name and retry. If unsuccessful, touch SF'15 and run the Sort Disk Catalog utility to list the specified disk's catalog. Retry either using the correct reference file name or disk, or choose the create option. |
| FILE filename IS PROTECTED. | The file whose file name appears in the error message is a protected or scrambled program file and cannot be accessed by this utility. | Check if the correct disk was accessed and file name was specified. If incorrect, retry with correct parameters. Otherwise, skip this protected program file. |
| FILE-filename - LINE nnnn NOT FULLY DECOMPRESSED | The program file whose input file name appears in the error message has a line number (nnnn) containing a multistatement line which could not be decompressed because a unique line number could not be assigned to each statement. This is not an error message; it merely indicates where multistatement lines exist in the decompressed file due to a lack of available line numbers. | After the Utilities menu appears, the user may clear the station in use, load the decompressed program, renumber the program text, save the renumbered program to disk, and then decompress the program file. |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| File-filename - REM REFERENCED LINE nnnn | During Compression, a statement line containing a REM statement was deleted and a statement line elsewhere referenced the deleted line. | Examine a recent cross-reference listing or run the List/Cross-Reference utility to determine the statement line(s) referencing the deleted REM statement. Edit those statement lines referencing the deleted line and resave the program file. |
| FILE SPECIFIED IS NOT AN ISS-4 REFERENCE FILE. KEY RETURN(EXEC) | The reference file specified is a pre-ISS-4 reference file (old) and the type was specified as Old/Old or Old/New. | Touch the RETURN key. Retry with type set at Old/Old or Old/New or specify the correct file name or disk if incorrectly specified before. |
| FILE SPECIFIED IS NOT A PRE ISS-4 REFERENCE FILE. KEY RETURN(EXEC) | The reference file specified is not a pre-ISS-4 (new) reference file and the type was specified as Old/Old or Old/New. | Touch the RETURN key. Retry with the mode set at New/New or New/Old or specify the correct file name or disk if incorrectly specified before. |
| FILE SPECIFIED IS NOT A REFERENCE FILE. KEY RETURN(EXEC) TO RESUME? | The specified file name is not a reference file. It is, however, an active data file. | Check if the correct disk was accessed. Retry with the correct disk or file name. Run the Sort Disk Catalog utility to list the disk's catalog if unsure of the reference file file name. |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| FILE SPECIFIED WAS NOT CLOSED PROPERLY. | The reference file specified is either being accessed by another station or was accidentally left open during previous operations. | Check if another station is accessing the reference file specified. Either wait until that station is done and touch RETURN or touch SF'31 and perform other processing (retry later). If no station is accessing the specified file, touch SF'31 and run the File Status Report utility to close the reference file accidentally left open. |
| INACTIVE FILE | The specified file is not an active file at the specified disk address. | Touch the RETURN key. Reenter the correct file name, specify the correct disk address, or mount the correct disk. In the case of an accidentally scratched file, touch SF'31, run Alter Disk Index to change the file status from scratched to active, and retry. |
| INACTIVE NAME | When searching for a file name, the specified disk does not contain the specified file name. | Re-enter the correct file name. If the correct disk was not accessed, mount the correct disk or touch SF'31 to obtain the ISS Utilities menu. Retry, specifying the correct disk address. |
| INSUFFICIENT MEMORY FOR FULL INDEX LISTING. KEY RETURN(EXEC) TO RESUME? | Insufficient memory exists for this station to perform the requested operation. | To obtain a partial list, touch the RETURN key. Otherwise, touch SF'31. With a 2200MVP, retry after a larger partition size has been allocated to this station or, for any CPU |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| | | type, use a station with a larger memory size to perform this operation. |
| INPUT aaa ENDS AT LINE nnnn BUT INPUT bbb CONTINUES | The program file identified as INPUT ONE or INPUT TWO (e.g., aaa=ONE) ends at statement line nnnn, but the program file identified as INPUT TWO or INPUT ONE (e.g., bbb=TWO) has additional program text beyond statement nnnn. | None. This is an informative message. |
| INVALID RANGE | The entered upper limit of the file names is lower according to the 2200 ASCII collating sequence (character code assignments) than the entered lower limit of file names. | Touch the RETURN key and re-enter the upper limit correctly or touch SF'15. |
| LINE nnnn MISSING IN INPUT aaa | Statement line number nnnn exists in only one program file; it does not exist in the program file identified as INPUT ONE or INPUT TWO (e.g., aaa=TWO) | None. This is an informative message. |
| MOUNT message KEY RETURN (EXEC) TO RESUME? | The word "message" is replaced by INPUT DISK, ISS PLATTER, etc. The disk requested should be mounted. | Either mount the disk and touch the RETURN key, or touch SF'15 to change the address if incorrect, or touch SAF'31 to abort this utility. After touching the RETURN key, the reappearance of this prompt indicates the disk has not yet been mounted. In most cases, touch SF'31, reload the utility, and carefully enter the disk address. |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| nnnn BOTH FILES END | The two program files being compared both end at statement line number nnnn. | None. This is an informative message. |
| NAME ALREADY IN INDEX | When renaming a file, the specified file name already exists on this disk. | Re-enter a unique file name. Was the correct disk accessed? If not, touch SF'31 to obtain the ISS utilities menu and retry. |
| NEXT SECTOR NOT WRITTEN IN DC MODE (expanded print) | During a Data File Structure Dump, the next sector does not contain control bytes written by DATASAVE DC (or DA) statements; it cannot be dumped using the Data File Structure Dump option, but may be dumped using other dump options. | Retry, but dump the file or range of sectors using the Horizontal or Vertical dump options. |
| NOT AN ACTIVE FILE | The file specified is scratched or does not exist. | Only active files can be processed by this utility. Specify the correct file name. |
| PRINTER ALLOWS 'FILE' DUMP ONLY | The output device is specified as printer, and the input mode is specified as RANGE. | Touch the RETURN key. Change the output device to CRT or the input mode to file. |
| PRINTER REQUIRED KEY RETURN(EXEC) TO RESUME? | The utility chosen or the option (e.g., output device) of the utility program chosen requires a printer. The start-up printer address must be changed to an actual printer address (not blank or 005) in order to perform this function. | If a printer is currently available, touch SF'31 as required to obtain the start-up ENTER DESIRED OPTION change the printer address to reflect the availability of a printer, and retry. If a printer is not available, either touch SF'15 (if allowed by this utility) and choose |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| | | CRT output or wait until it is available, change the printer address (see above) and retry. |
| PROGRAMS COMPARE - ARE SAME | The two program files being compared are identical. | None. This is an informative message. |
| RANGE NOT SPECIFIED | With the input mode of RANGE, the sectors to dump is specified as ALL or has been omitted. | Touch the RETURN key. Specify the sector numbers to be dumped or change the input mode to File. |
| RE-ENTER | During entry of either a reference file name or an input file name when MODE=PART, the file name entered does not exist, is a scratched file, or is a program file instead of a reference file on the input disk. | Re-enter the correct input file name. Check if the correct disk is being accessed. Retry. If unsuccessful, run the Sort Disk Catalog utility to determine the correct file name and then retry. |
| RE-ENTER<br>nn xx / uu (ddd.) | During entry of a numeric value, the number entered (xx) falls outside of the acceptable upper and lower numeric limits (range) which are displayed. The (ddd.) indicates the number of digits and decimal point position. | Re-enter a numeric value within the displayed upper (uu) and lower (nn) numeric limits. |
| RE-ENTER, DUPLICATE aaaaa FILE NAME. | The input or output (aaaaa) file name entered has already been entered. | Re-enter a correct file name. |
| RE-ENTER, FILE ALREADY EXISTS ON OUTPUT DISK | The specified file name already exists on the specified output disk. | Check if the correct disk has been specified. Re-enter the correct output file name or touch SF'15 to change the disk address or SF'31 to abort. |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| RE-ENTER, INPUT FIELD NOT DEFINED | The input file name has not yet been defined when attempting to enter the output file name. | Touch SF'0 and select an input file name. Then touch SF'1 and re-enter the output file name. |
| RE-ENTER, INVALID NAME | The output file entered was 0 (zero). | Enter a valid output file name. |
| RE-ENTER, INVALID RANGE | The entered upper limit of of file names is lower than the entered lower limit of file names, according to the 2200 ASCII collating sequence. | See INVALID RANGE; however, the RETURN key need not be touched. |
| RE-ENTER, MODE AND ADDRESSES INCOMPATIBLE | The specified mode was ALL, DATA, PROGRAM, or RANGE and the input and output disk addresses are the same. | Change the mode or change either the input or output disk address. |
| RE-ENTER, NAMES AND ADDRESSES INCOMPATIBLE | The input and output file names are identical when the same disk was specified for the input and output disk. | Change the output file name and/or disk addresses. Retry. |
| RE-ENTER, NOT AN ACTIVE FILE | The file specified is scratched or does not exist. | Only active files can be processed by this utility. Specify the correct file name. |
| RE-ENTER, NOT AN ACTIVE FILE ON aaaaa DISK. | The file whose file name was specified does not exist or is a scratched file on the input or output (aaaaa) disk accessed. | Enter the correct file name. |
| RE-ENTER, OUTPUT FIELD NOT DEFINED | The input and/or output file name has not yet been defined when attempting to enter an extra sectors value. | Touch the LINE ERASE key and then touch SF'0. Select an input file name. Touch SF'1 and re-enter the output file name and extra sectors value. |

Table E-1. ISS Utilities Error Messages and Recovery Procedures (continued)

| ERROR MESSAGE | DESCRIPTION | RECOVERY |
|---|---|---|
| RE-ENTER, WRONG FILE TYPE. | The file whose file name was entered is a program file when only data files are acceptable or a data file when only program files are acceptable. | Enter the correct file name. |
| RE-ENTER, WRONG PASSWORD | The password entered is not valid for this file. Up to 16 characters are allowed per password. | Re-enter the correct password. |
| RE-MOUNT ISS DISK AT xyy | The ISS disk has been removed from the ISS loading address (xyy). | Mount the ISS disk at the specified address and touch RETURN. |
| SECTORS MUST BE WITHIN DISK LIMITS | With the input mode of Range, the sector numbers specified for sectors to dump are outside to sector number boundaries of the specified disk. | Touch the RETURN key. Change the sector numbers to values contained within the specified disk's catalog area. |
| SECTORS MUST BE WITHIN FILE LIMITS | The specified sector numbers for the sectors to dump are not within the boundaries of the specified file. This applies to an output device of printer only. | Touch the RETURN key. Change the sector numbers to values contained within file boundaries. |
| SELECT PRINTER KEY RETURN(EXEC) TO RESUME? | The printer specified as the ISS start-up Printer Address is not powered on, not selected, or is currently being used by a different station. | Check if the printer is currently powered on, selected, or being used by another station. When ready, touch the RETURN key to continue. |
| TEXT DOES NOT MATCH ON LINE nnnn | Statement line number nnnn exists in both program files INPUT ONE and INPUT TWO, but their content differs. | None. This is an informative message. |

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.

700-5010A

TITLE OF MANUAL **INTEGRATED SUPPORT SYSTEM (ISS) RELEASE 5 USER MANUAL**

COMMENTS:

Fold

Fold

(Please tape, Postal regulations prohibit the use of staples.)

WANG

Fold

FIRST CLASS
PERMIT NO. 16
Lowell, 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