FOUNDATION DOCUMENT

A Technical Guide to the General program structure and File Handling Routines used by Peachtree Software[tm].

Revision 2.1
7/79

FOUNDATION DOCUMENT

TABLE OF CONTENTS

I. Introduction

This Foundation Document describes in detail the general structure and file handling routines used within the Peachtree Software^{tm} packages.

Although there are variations in programming techniques from package to package, this document serves to inform the user of the basic philosophies and mechanisms used specifically within the accounting, inventory packages, and other Peachtree packages.

Particular attention is given to the two ISAM file managers used within most Peachtree packages. The first of these managers, MARIS (Multi ARay ISam) is a single key ISAM written entirely in BASIC and appended to the end of each program. MARIS is used by each of the four accounting packages. The second file manager is MIKSAM (Multi Index Keyed Sequential Access Method). MIKSAM is incorporated within the Inventory, Timekeeping, and Mailing Address packages. MIKSAM is a sophisticated multi-keyed ISAM supporting variable length records. It is written in assembly code (for effeciency) and interfaced to the BASIC packages through subroutine calls.

Necessary adjuncts to the Foundation Document are the individually packaged system documentation which describe each of the application packages. Each of these documents narrate the particular programs which comprise the application system along with the files used and reports produced by each program.

II. Hardware/Software Requirements

Hardware Requirements
- Any 8080-type microcomputer
- 132 column printer
- CRT Terminal (24 lines, 80 characters per line)
- 2 flexible disk units
- 48K bytes of RAM

Software Requirements
- Microsoft™ Disk BASIC language
- CP/M™ Operating System, or equivalent

The packages of Peachtree Software™ are designed to be virtually machine-independent; i.e., any proper hardware configuration which will fully support Microsoft BASIC should execute the Peachtree packages. Other than the programs which use the USR function to interface with the MIKSAM file routines, no device-dependent BASIC statements (PEEKS, POKES, IN, OUT, ETC.) are incorporated within the Peachtree programs.

III. Disk File Conventions

Peachtree Software packages are delivered on a single flexible diskette for sales, demo and customer training purposes. However, for general system use, two flexible drives and two diskettes are required. System programs reside on drive Ø/A while customer data resides on drive 1/B. Instructions are provided with each package to convert from a single-disk to a two-disk system.

There are two types of files on each package diskette-program files and data files. All program and data files are prefixed by a two-character code unique to each package; e.g., GL for the General Ledger package, PR for the Payroll package, and so on. All program files are suffixed by .BAS and all data files are suffixed by .DAT. For example, the Master File Maintenance program file in the General Ledger package is named GLMF.BAS. The General Ledger Master File itself is named GLMST.DAT.

The specific file names for each package are listed within that packages particular package document.

IV. Program BASIC Subroutine Map

Statement Range	Range Contents
Ø	Program file name, title, author's initial's, creation date, date last modified.
1	CLEAR required string space, copyright statement.
2	ON ERROR GOTO 545ØØ 'ERROR TRAP SETUP
3	PN$ = "Program Name" 'PROGRAM NAME ASSIGNMENT
4	GOSUB 5ØØØØ 'GET SYSTEM PARAMETERS FROM SYSTEM FILE
4ØØØØ - 49999	Package File Handler and Interface Routines.
5ØØØØ - 5Ø899	Get system info, date; verify diskette assignments; define functions.
5Ø9ØØ - 5Ø999	Print Heading to Screen.
51ØØØ - 51Ø99	Alphanumeric string input (Query String)
512ØØ - 51399	Numeric string input (Query Number)
514ØØ - 51499	Date string input (Query Date)
516ØØ - 51799	Dollar amount input (Query Dollar)
518ØØ - 51899	Yes - No input (Query Yes-No)
519ØØ - 51999	Top of page input (Query Page)
52ØØØ - 53999	(Not currently allocated)
54ØØØ - 54Ø49	INVALID ENTRY message subroutine
54Ø5Ø - 54Ø99	System error test subroutine
541ØØ - 54199	SYSTEM ERROR message subroutine
542ØØ - 54299	NUMBER OUT OF RANGE message subroutine
543ØØ - 54399	INCORRECT DATE message subroutine
544ØØ - 54499	(Not currently allocated)
545ØØ - 54599	BASIC Error Trap subroutine
546ØØ - 54699	SYSTEM File Assignment message subroutines
547ØØ - 59999	(Not currently allocated)
6ØØØØ - 655ØØ	MARIS or MIKSAM File Subroutines

SECTION V

PEACHTREE SOFTWARE

SKELETON

Open and Read System Parameter File

The system parameter file is opened and read. Drive "A" will be accessed first. If the file is not there, then the variables will be erased and MD$ will point to the appropriate drive.

```
50000 ON ERROR GOTO 50120:  BELL$=CHR$(7)+CHR$(7)+CHR$(7)
50010 OPEN "I",1,"A:GLSYS.DAT":INPUT #1,N$
50020 DIM V$(VAL(N$)):V$(0)=N$:FOR I=1 TO VAL(N$):INPUT#1,V$(I):NEXT I:CLOSE
50030 MD$=V$(11):SC$=V$(8):W=VAL(V$(7))
50050 IF MD$<>"A" AND MD$<>"B" THEN GOSUB 54650: GOTO 19900
50060 IF MD$="A" GOTO 50110
50070 D=N1:ERASE V$
50090 OPEN "I",1,MD$+":GLSYS.DAT":INPUT #1,N$
50095 DIM V$(VAL(N$)):V$(0)=N$:FORI=1TOVAL(N$):INPUT #1,V$(I):NEXT I:CLOSE
50100 MD$=V$(11): IF MD$<>"B" THEN GOSUB 54600: GOTO 19900
50110 ON ERROR GOTO 54500: GOTO 50160
50120 IF ERR=53 OR ERR=56 THEN GOSUB 54600: GOTO 19900
50130 IF ERR=54 OR ERR=62 OR ERR=65 THEN GOSUB 54650:GOTO 19900
50150 GOTO 54500
```

BELL$ - (CRT terminal alarm)
SC$ - (CRT terminal clear screen and home cursor command)
MD$ - Any system drive designator e.g. (B,C,D...)
W - (CRT terminal/printer width)
V$( ) will be defined within each system's internal documentation.

DEFINED FUNCTIONS

Initialize variables to perform pack double precision number into 4 byte string.

```
50160 N#=551903297536.5#:X9$=CHR$(0)+CHR$(0):X8$=CHR$(0)+CHR$(168):
      GOTO 50170
```

Subroutine to convert double precision number (DD#) into 4 byte string (II$).

```
50161 II$=MID$(MKD$(N#+DD#*100#),3,4):RETURN
```

Subroutine to convert 4 byte string (II$) into double precision number (DD#).

```
50165 DD#=(CVD(X9$+II$+X8$)-INT(N#))*.01#:RETURN
```

Defined function to convert 8 character date string "MM/DD/YY" into single precision number in reverse order YYMMDD and stripped of "/" is

```
50170 DEFFNSD!(S$)=VAL(RIGHT$(S$,2)+LEFT$(S$,2)+MID$(S$,4,2))
```

Defined function to convert single precision number into character string in the form of "MM/DD/YY".

```
50175 DEFFNDS$(D!)=MID$(STR$(D!),4,2)+"/"+MID$(STR$(D!),6)
      +"/"+MID$(STR$(D!),2,2)
```

Open System File Subroutine

Open system file and read DATE (VD$), System Name (VS), Company Name (VC). SC$=(CLEAR SCREEN + HOME CURSOR COMMAND).

```
50180 OPEN "I",3,"A:GLDA.DAT":INPUT #3,VD$:CLOSE 3   'DATE
50190 VS$=V$(1): VC$=V$(20)   'SYSTEM, COMPANY NAME
50195 FOR I=1 TO VAL(V$(23)): DK%(I)=VAL(V$(23+2*I)): NEXT
50200 PRINT SC$:GOSUB 50900:PRINT:PRINT   'DISPLAY HEADERS
50210 PRINT:PRINT"ONE MOMENT FOR PROGRAM STARTUP....":PRINT:RETURN
```

Print Heading

```
  W = WIDTH
VC$ = COMPANY NAME
VS$ = SYSTEM NAME
PN$ = PROGRAM NAME
VD$ = DATE
```

```
50900 PRINTTAB((W-LEN(VC$))/2);VC$:PRINTTAB((W-LEN(VS$))/2);VS$
50910 PRINTTAB((W-LEN(PN$))/2);PN$
50920 PRINTTAB((W-LEN(VD$))/2);VD$
50990 RETURN
```

Alphanumeric String Input (QUERY)

Q$ - The query prompting message or null ("") if no such message is to be given.

R$ - The user's response is first compared with "END" to test for program exit.

RL$ - The leftmost character of the user's response (R$).

This subroutine is used to query the user and obtain a response. The programmer provides the prompt in Q$, and R$ is returned with the user's response. Only Control C will thwart this subroutine.

```
51000 PRINT Q$;:LINE INPUT R$
51060 RL$=LEFT$(R$,1)
51065 IF R$="END" THEN GOTO 19900 ELSE RETURN
```

Numeric String Input (QUERY NUMBER)

Q$ - The prompting message or null ("") if no such message is to be given.

UB! - The upper bound allowed for N!

LB! - The lower bound allowed for N!

N! - The value returned by this subroutine in the range LB!<=N!<=UB!

This subroutine uses the Alphanumeric string input subroutine to prompt the user for a number. The input string R$ is checked to insure of a valid number. If R$ is null then N! = Ø. R$ is scanned for allowable blanks followed by an optional "+" or "-" followed by 1 or more digits in which there may be an optional ".". If R$ can be converted to a valid number, it's value is checked to see if it is in the range of LB! and UB!. If it is not in range, an "OUT OF RANGE" message is printed and a new number is requested. IF UB! = LB! then this range checking is omitted. No error conditions may arise.

```
51200 GOSUB51000:N!=0:IFR$=E$THENRETURNELSEDP=0:DG=0
51220 IFLEFT$(R$,1)=" "THENR$=MID$(R$,2):GOTO51220
51230 IFLEN(R$)=0GOTO51290
51240 FORJJ=1TOLEN(R$):CH=ASC(MID$(R$,JJ,1))
51250 IFCH>47ANDCH<58THENDG=1:GOTO51280
51260 IFCH=46THENDP=DP+1:IFDP>1THENGOSUB54000:GOTO51200ELSEGOTO51280
51270 IF(CH<>45ANDCH<>43)ORJJ<>1THENGOSUB54000:GOTO51200
51280 NEXTJJ:IFDG=0THENGOSUB54000:GOTO51200
51290 N!=VAL(R$):IF(N!>=LB!ANDN!<=UB!)ORLB!=UB!THENRETURN
51300 GOSUB54200:GOTO51200
```

Date String Input (QUERY DATE)

MO - The Resulting Month
DA - The Resulting Day
YR - The Resulting Year

This subroutine uses the Alphanumeric String Input Subroutine to prompt the user for a date. The string R$ returned from query is parsed to see if is a correct date, i.e., R$ is null in which case MO=DA=YR=0 or R$ is in the form MM/DD/YY where 1<=MM<=12 and 1<=DD<=31 and 50<=YY<=99. The slashes between MM, DD, and YY can be any symbol. R$ contains the string value from which MO, DA, and YR are derived.

```
51400 GOSUB 51000: IF LEN(R$)=0 THEN MO=0: DA=0: YR=0: RETURN
51420 IFLEN(R$)<6 OR LEN(R$)>8 GOTO 51450
51422 MO=INSTR(R$,"/"):IF MO=0 GOTO 51450 ELSE IF MO=2 THEN R$="0"+R$
51424 DA=INSTR(4,R$,"/"):IF DA=0 GOTO 51450
51426 IF DA=5 THEN RL$=LEFT$(R$,3): R$="0"+RIGHT$(R$,4):R$=RL$+R$
51430 MO=VAL(LEFT$(R$,2)):DA=VAL(MID$(R$,4,2)):YR=VAL(RIGHT$(R$,2))
51440 IF(MO>0)AND(MO<13)AND(DA>0)AND(DA<32)AND(YR>0)AND(YR<99)THENRETURN
51450 GOSUB54300:GOTO51400
```

Dollar Amount Input (QUERY DOLLAR)

D# - The dollar value returned

This subroutine uses the Alphanumeric String Input Subroutine to prompt the user for a value. The string R$ returned from query is parsed to see if it is a correct dollar value, i.e., R$ is null in which case D#=0 or R$ is a valid number (See QUERY NUMBER for number definition) or is a number preceeded by a "$". Any commas are ignored. No error conditions can arise. R$ contains the string value from which D# is derived.

```
51600 GOSUB51000:DP=0:DG=0
51620 IFLEFT$(R$,1)=" "THENR$=MID$(R$,2):GOTO51620
51640 JJ=INSTR(R$,",",")":IFJJ<>0THENR$=MID$(R$,1,JJ-1)+MID$(R$,JJ+1):
      GOTO51640
51650 IFLEFT$(R$,1)="$"THENR$=MID$(R$,2)
51655 IFLEN(R$)=0THEND#=0:RETURN
51660 FORJJ=1TOLEN(R$):CH=ASC(MID$(R$,JJ,1))
51670 IFCH>47ANDCH<58THENDG=1:GOTO51700
51680 IFCH=46THENDP=DP+1:IFDP>1THENGOSUB54000:GOTO51600ELSEGOTO51700
51690 IF(CH<>45)AND(CH<>43)ORJJ<>1THENGOSUB54000:GOTO51600
51700 NEXTJJ:IFDG=0THENGOSUB54000:GOTO51600
51710 D#=VAL(R$+"D"):RETURN
```

Yes-No Input (QUERY YES/NO)

```
51800 Q$=Q$+" (Y OR N)? "
51805 YES=0:GOSUB51000:IFRL$="Y"THENYES=-1:RETURN
51807 IF LEN(R$)=0 THEN RETURN
51810 IFRL$<>"N"THENGOSUB54000:GOTO51805 ELSE RETURN
51900 PRINT:PRINT"POSITION THE PAPER AT THE TOP OF A PAGE. "
51910 Q$="READY":GOSUB 51800: IF LEN(R$)=0 THEN RETURN
51920 IF NOT YES GOTO 51910 ELSE RETURN
```

The parameters Q$, Q, and QT are described in the query subroutine. Yes - set true (-1) if answer is yes, set false (0) if answer is no.

This subroutine uses the Alpha Numeric String Input Subroutine to prompt the user for "yes" or no". Only the leftmost character (RL$) is checked for a "Y" or "N". If neither of these occur, the question is reasked.

Basic Error Trap Location

```
54500 PRINT:PRINT"   BASIC ERROR NUMBER ";ERR;" LINE NO. ";ERL
54510 PRINT"   CONSULT BASIC MANUAL FOR EXPLANATION. "
54520 PRINT:PRINT"ABNORMAL END OF JOB ":PRINT
54530 FORI=1TO400:PRINTBELL$;:NEXT:RESUME 19900
```

Top-of-forms Subroutine

This subroutine will print from the current line (LC) to 65 and then set LC to Ø.

```
65529 FOR II=LC TO 65: LPRINT: NEXT II: LC=0: RETURN
```

Invalid Entry Subroutine

BELL$-CRT terminal audible alarm command

```
54000 PRINT"*** INVALID ENTRY.  PLEASE REENTER. ***";BELL$:RETURN
```

System Error Test

```
54050 IFER%=0THENRETURN
```

System Error Subroutine

```
54100 PRINT"*** SYSTEM ERROR ";ER%;". PLEASE CONSULT MANUAL. ***";BELL$
54110 GOTO 19900
```

Number Out of Range Subroutine

```
54200 PRINT"*** NUMBER IS OUT OF RANGE. ***";BELL$:RETURN
```

Incorrect Date Subroutine

```
54300 PRINT"*** INCORRECT DATE.  MUST BE FORM MM/DD/YY. ***";BELL$:RETURN
```

System File Not Found

```
54600 PRINT:PRINT "*** PROPER SYSTEM FILE NOT FOUND.
      CAN'T CONTINUE. ***":RETURN
```

System File Initialized Improperly

```
54650 PRINT:PRINT "*** SYSTEM FILE INITIALIZED IMPROPERLY.
      CAN'T CONTINUE. ***":RETURN
```

LIST VARIABLES FOR THE PEACHTREE SOFTWARE SKELETON

```
BELL$    -54000, 54100, 54200, 54300, 54530

CH       -51240, 51250, 51260, 51270, 51660, 51670, 51680, 51690

D!       -50175

D#       -51655, 51710

DA       -51400, 51424, 51426, 51430, 51440

DD#      -50161, 50165

DG       -51200, 51250, 51280, 51600, 51670, 51700

DK%(     -50195

DP       -51200, 51260, 51600, 51680

E$       -51200

ER%      -54050, 54100

I        -50020, 50095, 50195, 54530

II$      -50161, 50165

JJ       -51240, 51270, 51280, 51640, 51660, 51690, 51700

LB!      -51290

MD$      -50030, 50050, 50060, 50090, 50100

MO       -51400, 51422, 51430, 51440

N!       -51200, 51290

N#       -50160, 50161, 50165

N$       -50010, 50020, 50090, 50095

PN$      -50910

Q$       -51000, 51800, 51910

$        -51000, 51060, 51065, 51200, 51220, 51230, 51240, 51290, 51400
         -51420, 51422, 51424, 51426, 51430, 51620, 51640, 51650, 51655
         -51660, 51710, 51807, 51910
```

LIST VARIABLES FOR THE PEACHTREE SOFTWARE SKELETON

RL$	-51060, 51426, 51805, 51810
S$	-50170
SC$	-50030, 50200
UB!	-51290
V$	-50070
V$(	-50020, 50030, 50095, 50100, 50190, 50195
VC$	-50190, 50900
VD$	-50180, 50920
VS$	-50190, 50900
W	-50030, 50900, 50910, 50920
X8$	-50160, 50165
X9$	-50160, 50165
YES	-51805, 51920
YR	-51400, 51430, 51440

SECTION VI

MARIS

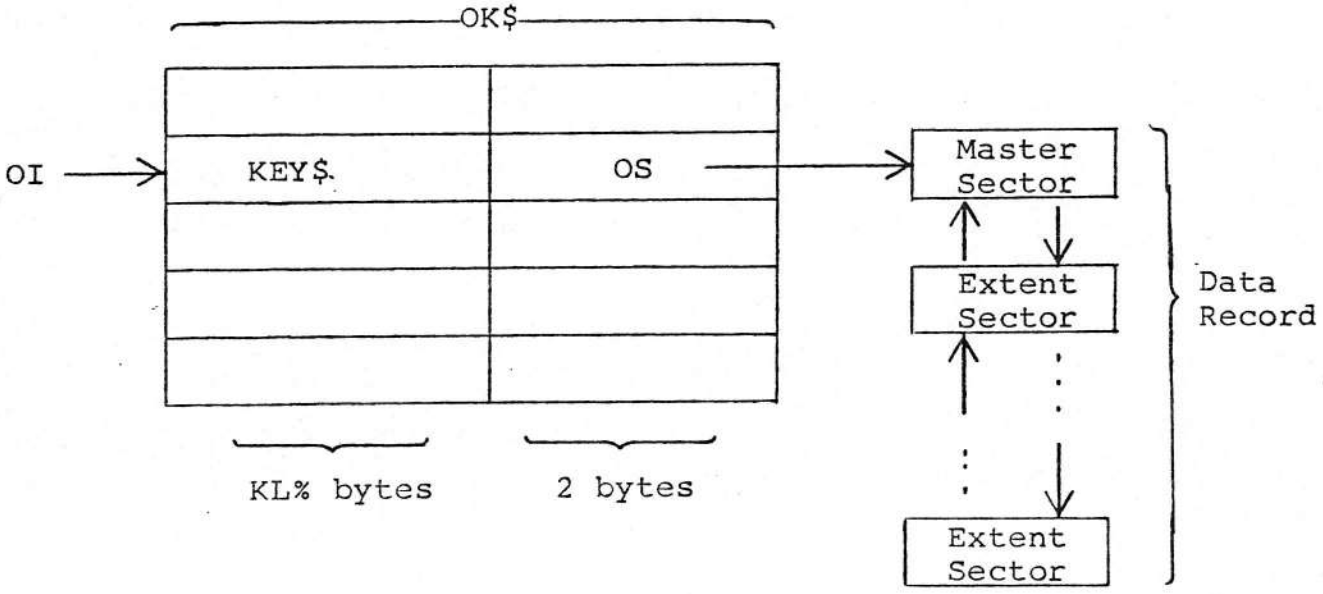
M A R I S

Multi ARray ISam

Steve Mann
(Revised 1/24/79)

MARIS OVERVIEW

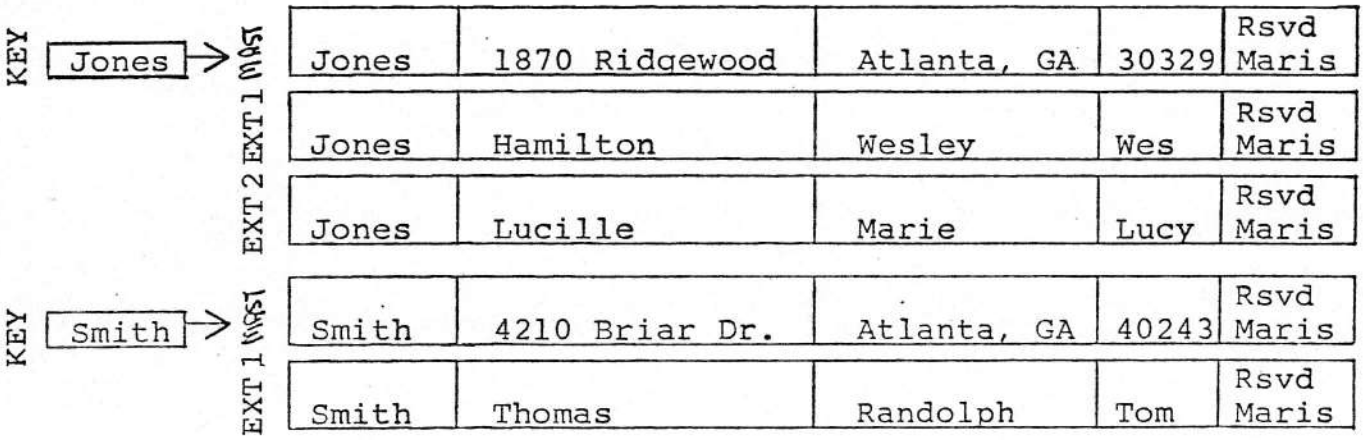
MARIS is a Multiple ARray Indexed Sequential method used to map Key values to record addresses in a file. In the diagram below, OI is the index which MARIS uses to point to the Key in array OK$. The Key is found using a binary search. Appended as the last two bytes of each Key is OS, which contains a pointer to the Master Sector. The Key is KL% bytes long, where KL% is defined by the programmer. Any element of the array OK$ is therefore KL% + 2 bytes long. The master sector can be linked to none, one, or more extent sectors, and together, they make up a data record.



To explain MARIS more effectively, the following example problem will serve as a guide for illustrating the interaction between a program and its file structure using MARIS.

Assume the tasks laid out before the programmer include maintaining a Membership Roster for the neighborhood Country Club. This roster must contain the names and address of each family belonging to the club. Furthermore, each family member who belongs to the Club must be kept on file. One way to arrange the file would be to allow the family last name to serve as a key for referencing: (1) the family street address, and (2) each family member's full name along with their "preferred" name.

A schematic representation of the file for the two records or families Jones and Smith might be:

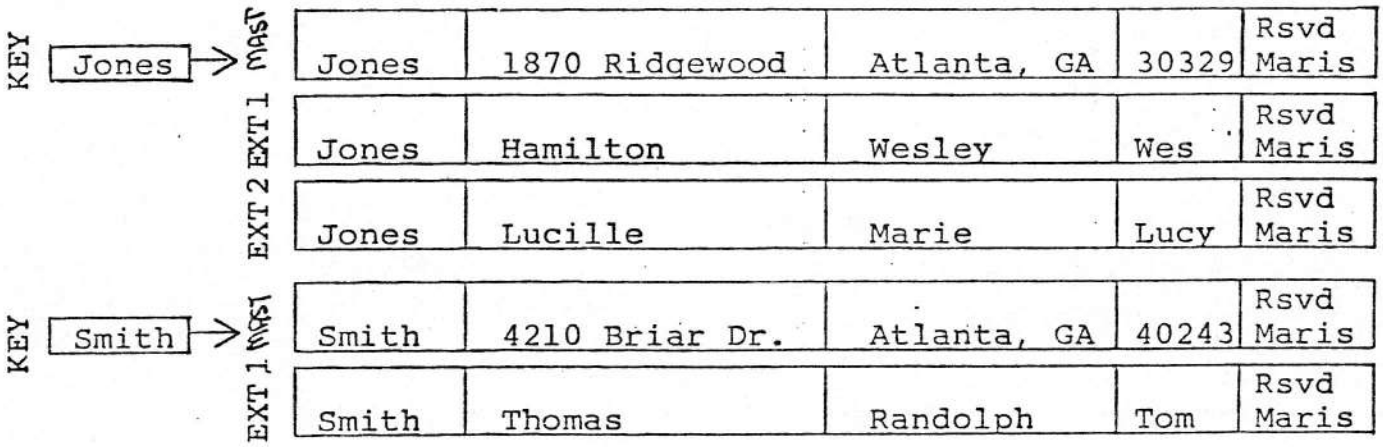


Notice that Jones and Smith can be used to find their respective families (they are keys). The Jones key points to the first record by specifically pointing to the sector which contains a duplicate of the key (Jones) and an address. MARIS refers to this first sector of the set as the Master Sector or simply as the Master. In this sector, a duplicate of the key was inserted by MARIS, while the programmer inserted the address information. The two sectors which extend downward from the Master Sector, are the sectors which hold each family member's name in the form of last name, first name, middle name, and preferred name. In MARIS, these sectors are referred to as Extent sectors. The collection of one Master Sector and (none, one, or more) Extent Sectors comprise a record. The last field associated with each sector, labeled RSVD MARIS show how the last five bytes of every sector are reserved for use by MARIS.

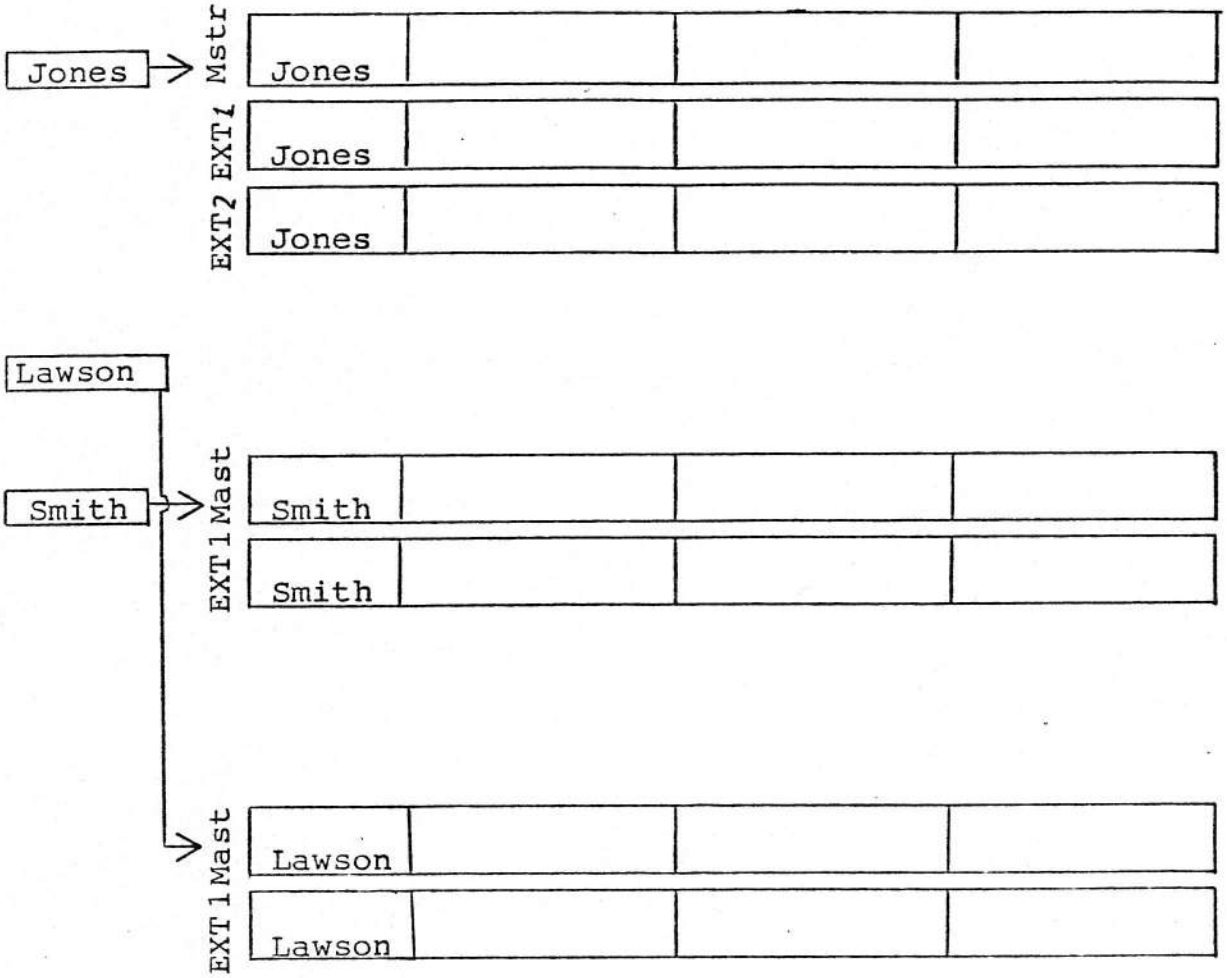
Add Family

The following is a schematic representation of adding a new Family to the Roster; and then deleting one. There is just one member in the newly joined Lawson Family. So a Master and one Extent are created for this record.

The file before addition:

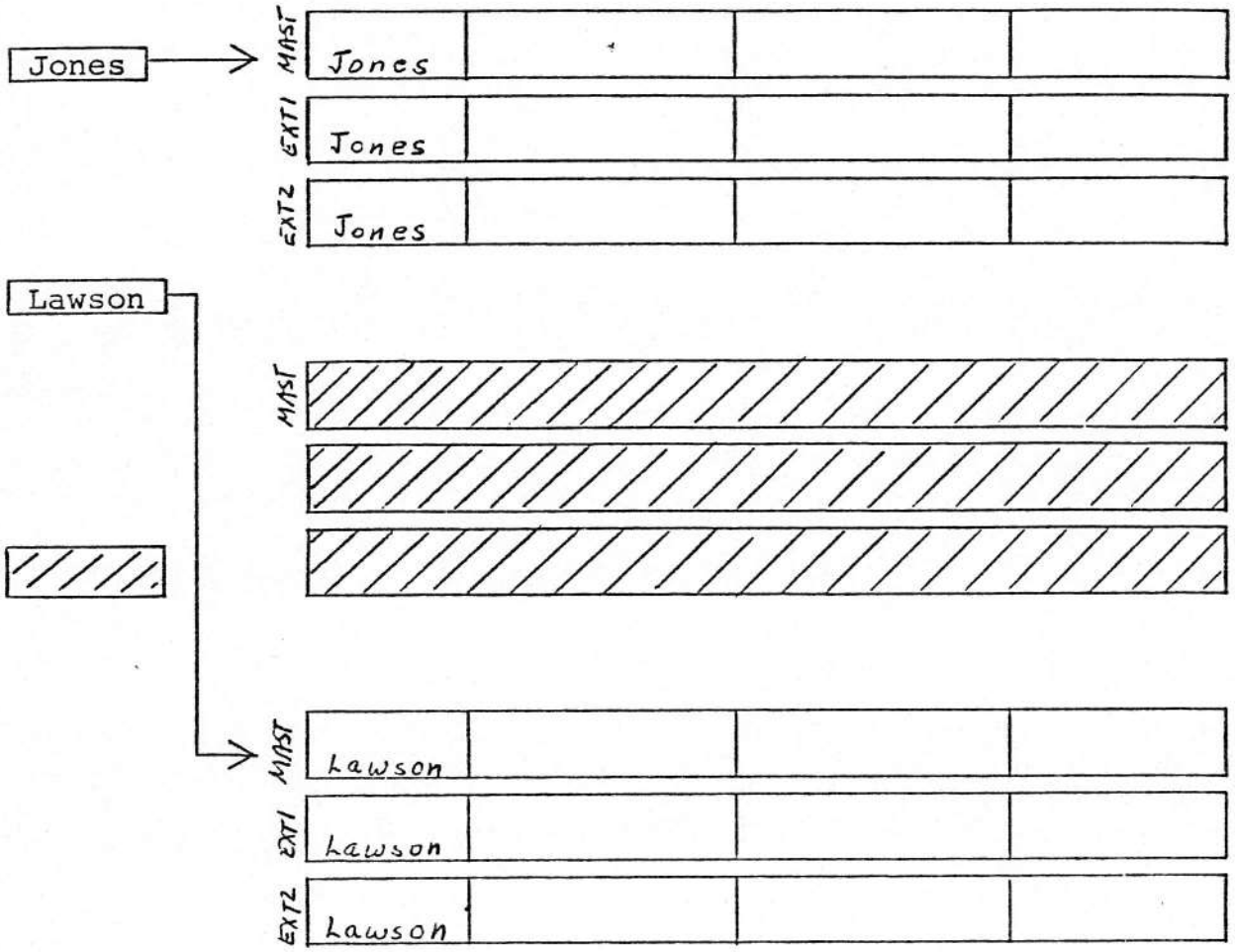


The file after addition:



Note how the keys are kept sorted in ascending order and point to the corresponding Master sector.

The file after deletion of the Smith Family.

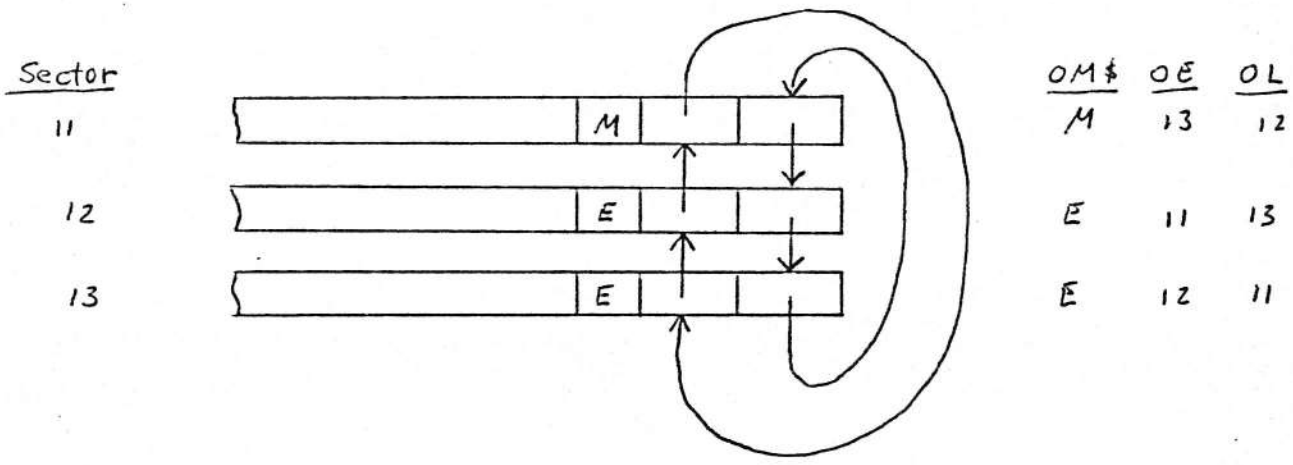


The shaded area which once contained the Smith family's record is now placed on an available list for later use by MARIS when new file space is needed.

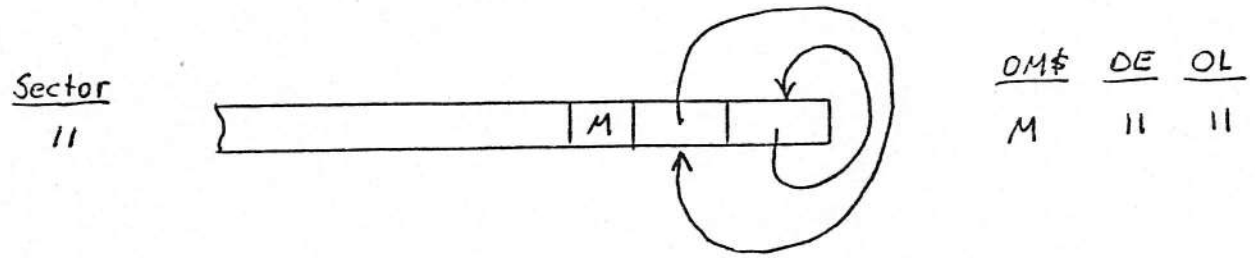
The following is a detailed look into the characteristics of a Master sector:

KEY	SECTOR DATA	OM$	OE	OL
KL% bytes	123-KL% bytes	1 byte	2 bytes	2 bytes

MARIS will LSET the key value into a file buffer having a size of KL% (key length). The sector data must be LSET by the programmer into a file buffer having a maximum size of 123-KL% bytes. The space difference between 128 bytes (the size of a sector) and 123, is the 5 bytes MARIS reserves for storing OM$, OE and OL. OM$ will contain "M" for Master sector, or "E" for Extent sector, or "F" to indicate the sector is "free" for re-use. OE and OL contain sector pointers. OE points to the sector location of the "next Earlier" ordered sector in the record, while OL points to the sector location of the "next Later" ordered sector in the record. In most operations, MARIS will return the value (OS), which is the current sector location.



Through the use of these pointers, a programmer can always determine the locations of the current sector (OS), and of its adjacent sectors (OE), (OL). Notice how MARIS closes a loop by having the MASTER (which has no Earlier sector) point to the last extent. By the same logic the last Extent (which has no later sector) points to the MASTER. When a record is first created, there are no Earlier or Later sectors to which the MASTER's (OE) and (OL) can point. So, they point to (OS), the sector number in which they reside. (OS) = (OE) = (OL)



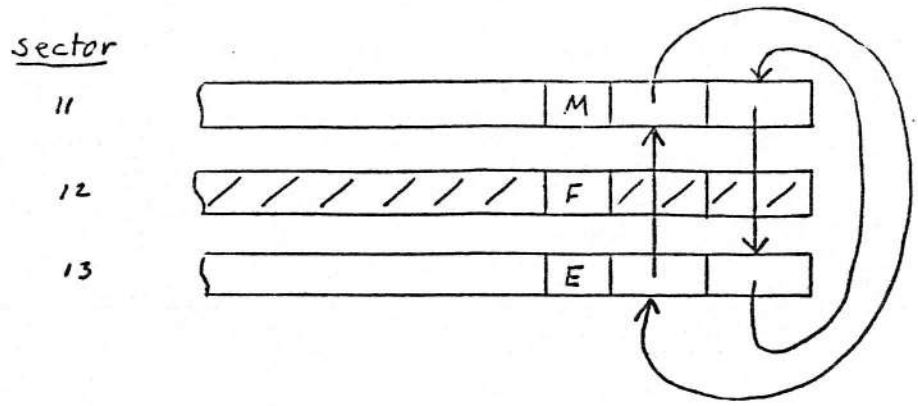
This also becomes true when all Extent sectors become deleted.

The Extent structure differs from the Master only in the fact that MARIS does not store a duplicate of the key value within the sector. However, it is advisable to maintain the key in each Extent (as shown in the examples), for recovery in the event that file integrity is lost.

An extent can be deletedfrom any record by specifying (OS), (the sector number of the extent to be deleted) and performing a DELETE ENTENT SECTOR function. MARIS will alter the adjacent sectors to point to each other: OE (pointer to next "earlier" sector) is set to the sector proceeding the deleted extent and OL)pointer to next later sector) is set to the sector following the deleted sector.

DELETE EXTENT

After first accessing the Master, its adjacent "Latest" record, (OL), becomes known. Setting OS to OL and performing a DELETE EXTENT SECTOR results in:

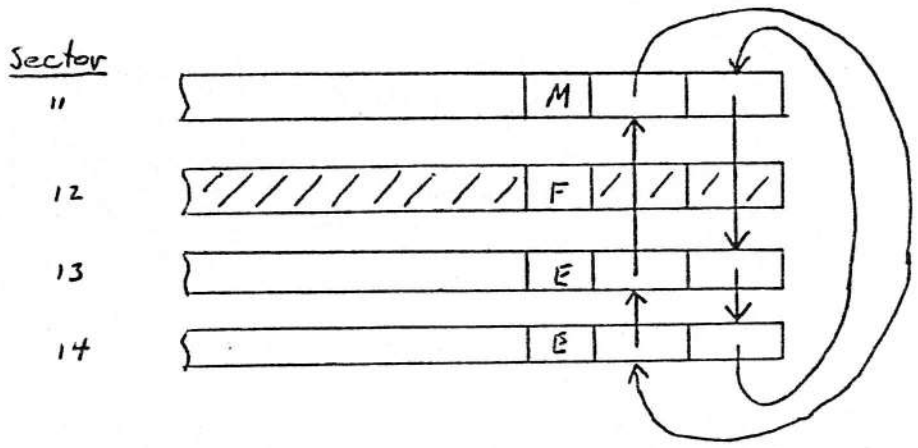


Extent 1 is deleted, and the two adjacent records now point to each other, Extent 2's (OE) now points to the Master and the Master's (OL) points to Extent 2.

To add a new Extent, pointers OE and OL must be set to the value the sector should have after creation.

ADD EXTENT 3

OE must be set to point to the new extent's predecessor, (OL) must be set to new extent's successor. To add Extent 3, (OE) must point to Extent 2, and (OL) will point to the Master. After first accessing the MASTER, the last Extent sector (pointed to by OE) becomes known. Setting OL to OS and leaving OE alone will arrange the pointers for the new extent (the new sector's OE field points to the old last extent and the OL field points to the master). Performing ADD EXTENT SECTOR will cause MARIS to calculate a new OS, and later the pointers of its adjacent records. The Extent is not actually saved until the programmer LSETS the appropriate fields and the statement "PUT FI%, OS" is performed.



Note that deletion of sector followed by creation of a sector without changing pointers results in an unchanged structure. Similarly, creation followed by deletion leaves the structure unchanged.

Should the last extent be deleted, the error code will be set to 1, implying that the end of the extent list has been

II. "MARIS" Operations

The following operation codes and their corresponding file procedures are described in order of operation number:

OP	FUNCTION PERFORMED
1	CREATE MARIS FILE (FI%, NA$, DI%, DA$, KL%, OC$, RETURNS: ER%)

The operation performed is to create a MARIS file with name NA$, on drive DI%, with creation date DA$, with key length KL%, and comment OC$. FI% is used as a temporary file number. ER% is set to 4 if the file already exists otherwise ER% is set to 0. The file is not opened after creation.

2 OPEN MARIS FILE (FI%, NA$, DI%, RETURNS: DA$, EX%, NR%, KL%, OC$, OI, ER%)

The file NA$ on drive DI% is opened as file number FI%. The variables DA$, EX%, NR%, OA, KL%, and OC$ are set to the creation date, the current extent of the file, the current number of records in the file, the key length, and the comment information respectively.

ER% is set to 5 if the file does not exist, to 1 if the file is empty, or to 0 if no error holds. If ER% = 0, a side effect of opening the file is that the file is rewound and OI is set to Ø. (The first index in the file.)

The key values and pointers are read from the file into an internal array for fast access. If the file had not been properly closed then this array was not saved in the file; the open function rebuilds the array by scanning the entire file and by sorting the keys. Hence, if the file was not properly closed, a longer time must pass before opening can be accomplished. If the file had not been closed properly, then the CLOSED flag will be set to 0, otherwise it will be set to 255. The CLOSED flag is in byte 6 of header record.

3 CLOSE MARIS FILE (RETURNS: ER%)

The MARIS file currently opened is closed and the key pointer array is written out at the end of the file. ER% is set to 6 if there is not enough room on the disk to hold these pointers. (Subsequent re-opening of the file will work properly although a longer time must pass. See Section V). ER% is set to 0 if the file is closed properly.

4 CREATE A MASTER RECORD (KE$, RETURNS: OI, OS, ER%)

The key/pointer array is searched for the key KE$. If found ER% is set to 3 and no action is performed; if the key is not found, one is created. OI is set to the key/pointer array index, OS is set to the master sector created, the buffer is set to receive information, and ER% is set 0. If ER% is set to 6 then the disk is out of room.

5 DELETE RECORD (KE$, RETURNS: KE$,OI, OS, ER%)

The key/pointer array is searched for the key KE$. If not found then ER% is set to 2 and no action is performed. If the key is found, the key is deleted from the array and the master and any extent sectors are placed on the available list for future use. If the end of the file is not reached, KE$ is set to the next key in the file, OI is set to the next array index, and OS is set to the sector of the next master record. ER% is set to 0 if no error occurs.

6 REWIND (RETURNS: OI, ER%)

The file is rewound so that the master sector of the first logical record in the file is the next record to be read. If the file is empty, ER% is set to 1, otherwise ER% is set to Ø. OI is set to Ø.

7 SEARCH (KE$, RETURNS: OI, OS, ER%)

The key/pointer array is searched for the key KE$. If found then ER% is set to 3, OI is set to the array index, and OS is set to the sector containing the master sector of the corresponding record. If the key KE$ is not found then ER% is set to 2.

8 GET RECORD (KE$, RETURNS: OI, OS, OE, OL, ER%)

The file is searched for a record with key KE$. If not found then ER% is set to 2 and no record is obtained. If ER% is zero then the record was found; OI is set to the array index, OS is set to the sector of the master area, OE is set to link to the end

OP FUNCTION PERFORMED (Cont.)

extent, OL is set to link to the first extent, and the buffer holds the master sector of the record.

9 GET NEXT RECORD (RETURNS: KE$, OI, OS, OE, OL, ER%)

The next logical record in the file is obtained. If ER% = 0, then KE$ holds the key of the next record, OI holds the array index, OS holds the sector number of the master sector, OE is the link to the end extent, OL is the link to the first extent, and the buffer holds the master sector. If ER% = 1, then the end of the file was reached.

10 GET PREVIOUS RECORD (RETURNS: KE$, OI, OS, OE, OL, ER%)

The record preceding the current logical record in the file is obtained. If ER% = 1 then there is no preceding record (i.e., the front of the file was reached). If ER% = 0 then KE$ holds the key of the previous record, OI holds the array index, OS holds the sector number of the master sector, OE is the link to the first extent, and the buffer holds the master sector of the previous record.

11 CREATE EXTENT SECTOR (OE, OL, RETURNS: OS, ER%)

If ER% = 0, a new extent record is created. OS is changed to point to the new extent sector. The earlier extent field in the new extent is set to OE, the later extent field in the new extent is set to OL, and the buffer is set to receive information. ER% is set to 6 if no room is available on the disk.

12 DELETE EXTENT SECTOR (OS RETURNS: OE, OL, ER%)

If ER% = Ø , then an existing extent sector OS is deleted. The result of this operation is that OE is set to the earlier extent before the deleted extent OL is set to the later extent after the deleted extent and the deleted sector is placed on the available list for future use. IF ER% =1, then the end of the extent has been reached.

13 GET NEXTENT SECTOR (OS, RETRUNS: OE, OL, ER%)

If ER% = 0), then the sector OS is obtained. OE is set to the earlier extent, OL is set to the later extent, and the buffer is set to hold the next extent sector. If ER% = 1, then the end of extents has been reached.

OP FUNCTION PERFORMED (Cont.)

PUT MASTER OR EXTENT RECORD (OS):

There is no OP subroutine to write the record contents of a sector; rather the programmer should use the following BASIC statement:

```
PUT FI%, OS
```

Where FI% is the file number and OS contains the sector number, usually determined by MARIS.

III. Variables and Their Use

Name	USE
DA$	File creation date
DI%	Disk drive file is mounted on
ER%	Error indication
EX%	Extent of file
FI%	File number
KE$	Key value
KL%	Key length
NA$	Name of file
NR%	Number of records in the file
0Ø-07	Header field descriptions for "MARIS", CLOSED FLAG, DA$, EX%, NR%, OA KL%, and OC$ respectively.
OA	Available sector
OB	Temporary - indicates bottom of search list
OC$	User comment
OD	Drive file is mounted on
OE	Link to earlier extent
OE$	Field for earlier extent pointer.
OI	Index in key array - indicates current position
OJ	Temporary
OK	Temporary
OK$	Field for key value
OK$()	Key array index and pointer values
OL	Link value to later extent
OL$	Field for later extent pointer
OM$	Field to indicate sector type: M= Master, E = Extent, F = Free.
OP	Operation to be performed in MARIS system
OS	Sector Value
OT	Temporary
OT$	Temporary

IV. Error Numbers and OP Codes

The following error numbers are also condition numbers, usually ER% = 1 is not an error.

ER%	Meaning	OP Codes of routines where error can occur
0	No error	All routines
1	End of file	2, 6, 9, 10, 5
1	End of extent	12,13
2	Key not found	5,7,8
3	Key found	4,7
4	File already exists	1
5	File does not exist	2
6	Out of space	3,4,11
7	Invalid OP code	

OP	MARIS Function Performed
1	CREATE MARIS FILE (FI%, NA$, DI%, DA$, KL%, OC$, RETURNS: ER%)
2	OPEN MARIS FILE (FI%, NA$, DI%, RETURNS: DA$, EX%, NR%, KL%, OC$, OI, ER%)
3	CLOSE MARIS FILE (RETURNS: ER%)
4	CREATE RECORD (KE$, RETURNS: OI, OS, ER%)
5	DELETE RECORD (KE$, RETURNS: KE$, OI, OS, ER%)
6	REWIND (RETURNS: OI, ER%)
7	SEARCH (KE$, RETURNS: OI, OS, ER%)
8	GET RECORD (KE$, RETURNS: OI, OS. OE, OL, ER%)
9	GET NEXT RECORD (RETURNS: KE$, OI, OS, OE, OL, ER%)
10	GET PREVIOUS RECORD (RETURNS: KE$, OI, OS, OE, OL, ER%)
11	CREATE EXTENT SECTOR (OE, OL, RETURNS: OS, ER%)
12	DELETE EXTENT SECTOR (OS, RETURNS: OE, OL, ER%)
13	GET EXTENT SECTOR (OS, RETRUNS: OE, OL, ER%)

GOSUB 62000 SETUP (NR%)

V. General Comments

Should out-of space occur, no data is lost except the last record entered or the key/pointer array. Deletions from the file will work properly and new additons can be added up to the number of sectors deleted. If out-of-space occurs while closing the file and writing out the array, the file cannot be closed properly even though deletions may have been performed. This is because the array is saved at the physical end of the file and logical deletions do not allocate physical space. The file should be backed up (End-of-Period should be run) and the resulting new file will be packed (i.e., all unused space will be deallocated). A file closed improperly can be opened although this will require time to rebuild the key/pointer array.

Traversing a record (obtaining the master and all extent sectors) may be accomplished in the following fashion:

```
100 GET RECORD (KE$, RETURNS: OI, OS, OE, OL, ER%)
110 IF ER% ≠ Ø THEN GO TO 999 'record was not found
    'operate on master sector
200 OS = OL: GET EXTENT SECTOR (OS, RETURNS: OE, OL, ER%)
210 IF ER% ≠ 0 THEN GO TO 999 ' No link sector found
    'operate on master sector
300 GO TO 200
999 'record was traversed
```

Traversing a file (obtaining the first and all succeeding records in a file) is accomplished by:

```
100 REWIND (RETURNS: OI, ER%)
110 GET NEXT RECORD (RETRUNS: KE$, OI, OS, OE, OL, ER%)
120 IF ER  ≠  Ø THEN GO TO 999 'end of file reached
    'operate on record
200 GO TO 110
999 'file was traversed
```

Putting information in a sector requires first obtaining or creating the sector, LSETing information into the sector, and saving the sector.

To create information in a NEW master sector:

```
100 CREATE RECORD (KE$, RETURNS: OI, OS, OE, OL, ER%)
    'LSET information into the buffer if ER% = 0
200 PUT FI%, OS'save the sector out
```

```
    To update information in an EXISTING extent:
100 GET RECORD (KE$, RETURNS: OI, OS, OE, OL, ER%)
    'LSET information into the buffer if ER% = 0

200 PUT FI%, OS 'save the sector out
```

All programs that use the MARIS file system must allocate enough string space to handle program needs and key/pointer array space. If the key length is KL% and the maximum number of records in the file is NR%, then there must be at least

```
(KL% + 5) * NR% + C
```

bytes allocated for string space. C is a constant dictated by the program needs; usually 1000 to 3000 bytes is enough. For example, if there are at most 300 records with key length of 6 bytes, then (6+5) * 300 = 3300 bytes are needed for the key/pointer array. A CLEAR 5000 should suffice for program needs.

COMPARISONS BETWEEN MARIS AND ISAM

The MARIS file system is a replacement for the existing ISAM file system whenever the file structure to be implemented consists of a set of records with each record being comprised of a master area and a variable number of extent areas. Thus the General Ledger, Accounts Receivable, Accounts Payable, and Payroll Systems can be implemented with a MARIS file system. For example, the General Ledger in ISAM consists of two indexed files, the MASTER file and the JOURNAL ENTRIES file. In MARIS these two files become one file, the master area holds the MASTER account record and the extent area holds the JOURNAL ENTRIES records.

Unlike ISAM, all MARIS calls are GOSUB 60000 with an operation code, OP, set to a value between 1 and 13 depending on what operation is to be performed. The only exception to this is the SETUP MARIS call. This is one by setting NR% to the maximum number of records and by performing a GOSUB 62000. This sets up the key/pointer array space.

LIST OF VARIABLES FOR MARIS

OE	-60800, 60900, 61000, 61100, 61210, 61300, 62200
OE$	-60250, 60420, 60800, 60900, 61000, 61100, 61210, 61300, 62200
OH	-60283, 60284, 60285
OH(	-60281, 60282, 60283
OI	-60260, 60270, 60283, 60284, 60285, 60310, 60320, 60420, 60500 -60520, 60600, 60710, 60720, 60730, 60900, 61000
OJ	-60260, 60270, 60283, 60285, 60310, 60320, 60410, 60500
OK	-60260, 60283, 60285, 60308, 60310
OK$	-60250, 60280, 60420, 60900, 61000
OK$(	-60270, 60280, 60283, 60284, 60285, 60310, 60410, 60420, 60500 -60520, 60710, 60730, 60900, 61000, 62000
OL	-60510, 60800, 60900, 61000, 61100, 61210, 61300, 62200
OL$	-60250, 60420, 60800, 60900, 61000, 61100, 61210, 61300, 62100 -62200
OM$	-60250, 60280, 60420, 61100, 61200, 61300, 62200
OP	-60000, 60280, 60281
OS	-60420, 60500, 60510, 60520, 60730, 60800, 60900, 61000, 61100 -61200, 61300, 62100, 62200
OT	-60260, 60281, 60282, 60283, 60308, 60310, 60500, 60510, 60700 -60710, 60720
OT$	-60250, 60260, 60270, 60283, 60284, 60285, 60310, 60320, 60710 -60720
OT$(	-60260, 60270, 60310

LIST OF VARIABLES FOR MARIS

BE$	-60280
DA$	-60140, 60250
DI%	-60100, 60200, 60290
ER%	-60090, 60110, 60120, 60150, 60210, 60220, 60240, 60300, 60302 -60304, 60330, 60400, 60420, 60500, 60520, 60600, 60710, 60730 -60800, 60900, 61000, 61100, 61200, 61210, 61300, 62110
EX%	-60250, 60260, 60280, 60310, 60330, 62100, 62110
FI%	-60130, 60150, 60230, 60240, 60250, 60260, 60280, 60290, 60300 -60310, 60320, 60330, 60420, 60800, 60900, 61000, 61100, 61200 -61210, 61300, 62100, 62110, 62200
KE$	-60420, 60520, 60700, 60720, 60900, 61000
KL%	-60140, 60250, 60260, 60308, 60520, 60700, 60710
NA$	-60100, 60130, 60200, 60230
NR%	-60250, 60260, 60280, 60281, 60282, 60283, 60308, 60310, 60330 -60410, 60420, 60500, 60520, 60600, 60700, 60900, 62000
O	-62000
O0$	-60130, 60140, 60230, 60240
O1$	-60130, 60140, 60230, 60250, 60290, 60330
O2$	-60130, 60140, 60230, 60250
O3$	-60130, 60140, 60230, 60250, 60330, 62110
O4$	-60130, 60140, 60230, 60250, 60330
O5$	-60130, 60140, 60230, 60250, 60330, 62110
O6$	-60130, 60140, 60230, 60250
O7$	-60130, 60140, 60230, 60250, 60330
O8$	-60130, 60140
OA	-60250, 60330, 62100, 62110, 62200
OB	-60260, 60310, 60410, 60420, 60700, 60710, 60720
OC$	-60140, 60250, 60330
OD	-60290

```
999 '10/30/78
                MULTI  ARRAY  ISAM  -  MARIS (OP)

000 ONERRORGOTO60090:ONOPGOSUB60100,60200,60300,60400,60500,60600,60700,
      60800,60900,61000,61100,61200,61300
010 ONERRORGOTO54500:RETURN
090 IFERR=61THENER%=6:RESUME60010ELSEONERRORGOTO54500:RESUME54500
099 '
                CREATE MARIS FILE (FI%, NA$, DI%, DA$, KL%, OC$, RETURNS: ER%)

100 ONERRORGOTO60110:NA$=CHR$(DI%+65)+":"+NA$:NAMENA$ASNA$
110 ER%=ERR:RESUME60120
120 ONERRORGOTO60090:IFER%=58THENER%=4:RETURN
130 OPEN"R",FI%,NA$:NA$=MID$(NA$,3):
    FIELDFI%,5ASO0$,1ASO1$,8ASO2$,2ASO3$,2ASO4$,2ASO5$,2ASO6$,100ASO7$,6ASO8$
140 LSETO0$="MARIS":LSETO1$=CHR$(255):LSETO2$=DA$:LSETO3$=MKI$(1):
   LSETO4$=MKI$(0):LSETO5$=O4$:LSETO6$=MKI$(KL%):LSETO7$=OC$:LSETO8$="103078"
150 PUTFI%,1:CLOSEFI%:ER%=0:RETURN
199 '
                OPEN MARIS FILE (FI%, NA$, DI%,
                                 RETURNS: DA$, EX%, NR%, OA, KL%, OC$, OI, ER%)

210 ONERRORGOTO60210:NA$=CHR$(DI%+65)+":"+NA$:NAMENA$ASNA$
220 ER%=ERR:RESUME60220
230 ONERRORGOTO60090:IFER%<>58THENER%=5:RETURN
    OPEN"R",FI%,NA$:NA$=MID$(NA$,3):
      FIELDFI%,5ASO0$,1ASO1$,8ASO2$,2ASO3$,2ASO4$,2ASO5$,2ASO6$,100ASO7$
240 GETFI%,1:IFO0$<>"MARIS"THENCLOSEFI%:ER%=5:RETURN
250 DA$=O2$:EX%=CVI(O3$):NR%=CVI(O4$):OA=CVI(O5$):KL%=CVI(O6$):OC$=O7$:
      FIELDFI%,(KL%)ASOK$,(123-KL%)ASOT$,1ASOM$,2ASOE$,2ASOL$:
      IFO1$<>CHR$(255)THENO6028OELSEIFNR%=0THENO6029O
260 OK=KL%+2:OT=128\OK-1:OJ=OT+1:GETFI%,EX%:DIMOT$(OT):OB=0:
      FOROI=0TOOT:FIELDFI%,(OB)ASOT$,(OK)ASOT$(OI):OB=OB+OK:NEXT:
      FOROI=1TONR%:IFOJ>OTTHENOJ=0:GETFI%
270 OK$(OI)=OT$(OJ):OJ=OJ+1:NEXTOI:ERASEOT$:GOTO60290
280 PRINT:PRINT"*** FILE WAS NOT CLOSED PROPERLY.  ONE MOMENT FOR FIX...":
      PRINTBE$:NR%=0:GETFI%,1:IFEX%<2THEN60290ELSEFOROP=2TOEX%:GETFI%:
      IFOM$="M"THENNR%=NR%+1:OK$(NR%)=OK$+MKI$(OP)
281 NEXTOP:IFNR%<2THEN60290ELSEDIMOH(9):OH(1)=1:OH(2)=4:OH(3)=13:OT=1
282 IFOH(OT+2)<NR%THENOT=OT+1:OH(OT+2)=3*OH(OT+1)+1:GOTO60282
283 FOROK=OTTO1STEP-1:OH=OH(OK):FOROJ=OH+1TONR%:OI=OJ-OH:SWAPOT$,OK$(OJ)
284 IFOT$<OK$(OI)THENSWAPOK$(OI+OH),OK$(OI):OI=OI-OH:IFOI>0THEN60284
285 SWAPOK$(OI+OH),OT$:NEXTOJ,OK:ERASEOH
290 GETFI%,1:LSETO1$=CHR$(0):PUTFI%,1:OD=DI%:GOTO60600
```

```
60299 '
                CLOSE MARIS FILE (RETURNS: ER%)

60300 ONERRORGOTO60302:GETFI%,1:ER%=4:GOTO60304
60302 ER%=0:RESUME60304
60304 ONERRORGOTO60090:IFER%=0THENRETURN
60308 IFNR%=0THEN60330ELSEOK=KL%+2:OT=128\OK-1
60310 OJ=0:GETFI%,EX%:DIMOT$(OT):OB=0:
        FOROI=0TOOT:FIELDFI%,(OB)ASOT$,(OK)ASOT$(OI):OB=OB+OK:NEXT:
        FOROI=1TONR%:LSETOT$(OJ)=OK$(OI):OJ=OJ+1:IFOJ>OTTHENPUTFI%:OJ=0
60320 NEXTOI:ERASEOT$:IFOJ<>0THENPUTFI%
60330 GETFI%,1:LSETO1$=CHR$(255):LSETO3$=MKI$(EX%):LSETO4$=MKI$(NR%):
        LSETO5$=MKI$(OA):LSETO7$=OC$:PUTFI%,1:CLOSEFI%:ER%=0:RETURN
60399 '
                CREATE RECORD (KE$, RETURNS: OI, OS, ER%)

60400 GOSUB60700:IFER%=3THENRETURNELSEGOSUB62100
60410 IFNR%>=OBTHENFOROJ=NR%TOOBSTEP-1:SWAPOK$(OJ+1),OK$(OJ):NEXT
60420 NR%=NR%+1:OI=OB:OK$(OI)=KE$+MKI$(OS):LSETOK$=KE$:LSETOM$="M":
        LSETOE$=MKI$(OS):LSETOL$=MKI$(OS):PUTFI%,OS:ER%=0:RETURN
60499 '
                DELETE RECORD (KE$, RETURNS: KE$, OI, OS, ER%)

60500 GOSUB60700:IFER%=2THENRETURNELSEGOSUB62200:NR%=NR%-1:OT=OS:
        IFOI<=NR%THENFOROJ=OITONR%:SWAPOK$(OJ),OK$(OJ+1):NEXT
60510 OS=OL:IFOS<>OTTHENGOSUB62200:GOTO60510
60520 IFOI>NR%THENER%=1:RETURNELSEOS=CVI(RIGHT$(OK$(OI),2)):
        KE$=LEFT$(OK$(OI),KL%):ER%=0:RETURN
60599 '
                REWIND (RETURNS: OI, ER%)

60600 OI=0:IFNR%=0THENER%=1:RETURNELSEER%=0:RETURN
60699 '
                SEARCH (KE$, RETURNS: OI, OS, ER%)

60700 OT=NR%:OB=1:IFLEN(KE$)<>KL%THENKE$=LEFT$(KE$+STRING$(KL%," "),KL%)
60710 IFOB<=OTTHENOI=(OT+OB)\2:OT$=LEFT$(OK$(OI),KL%)ELSEER%=2:RETURN
60720 IFOT$>KE$THENOT=OI-1:GOTO60710ELSEIFOT$<KE$THENOB=OI+1:GOTO60710
60730 OS=CVI(RIGHT$(OK$(OI),2)):ER%=3:RETURN
60799 '
                GET RECORD (KE$, RETURNS: OI, OS, OE, OL, ER%)

60800 GOSUB60700:IFER%=2THENRETURN
        ELSEGETFI%,OS:OE=CVI(OE$):OL=CVI(OL$):ER%=0:RETURN
60899 '
                GET NEXT RECORD (RETURNS: KE$, OI, OS, OE, OL, ER%)

60900 IFOI>=NR%THENER%=1:RETURNELSEOI=OI+1:OS=CVI(RIGHT$(OK$(OI),2)):GETFI%,OS:
        KE$=OK$:OE=CVI(OE$):OL=CVI(OL$):ER%=0:RETURN
```

```
1999 '
                GET PREVIOUS RECORD (RETURNS: KE$, OI, OS, OE, OL, ER%)

2000 IFOI<=1THENER%=1:RETURNELSEOI=OI-1:OS=CVI(RIGHT$(OK$(OI),2)):GETFI%,OS:
       KE$=OK$:OE=CVI(OE$):OL=CVI(OL$):ER%=0:RETURN
2099 '
                CREATE EXTENT SECTOR (OE, OL, RETURNS: OS, ER%)

2100 GOSUB62100:GETFI%,OE:LSETOL$=MKI$(OS):PUTFI%,OE:
       GETFI%,OL:LSETOE$=MKI$(OS):PUTFI%,OL:GETFI%,OS:LSETOE$=MKI$(OE):
       LSETOL$=MKI$(OL):LSETOM$="E":PUTFI%,OS:ER%=0:RETURN
2199 '
                DELETE EXTENT SECTOR (OS, RETURNS: OE, OL, ER%)

2200 GETFI%,OS:IFOM$<>"E"THENER%=1:RETURNELSEGOSUB62200
2210 GETFI%,OE:LSETOL$=MKI$(OL):PUTFI%,OE:
       GETFI%,OL:LSETOE$=MKI$(OE):PUTFI%,OL:ER%=0:RETURN
2299 '
                GET EXTENT SECTOR (OS, RETURNS: OE, OL, ER%)

2300 GETFI%,OS:IFOM$<>"E"THENER%=1ELSEOE=CVI(OE$):OL=CVI(OL$):ER%=0
2310 RETURN
2399 '
                SETUP (NR%)

2000 DEFINTO:DIMOK$(NR%):RETURN
2099 '
                GET NEW SECTOR (RETURNS: OS)

2100 IFOA<>0THENGETFI%,OA:OS=OA:OA=CVI(OL$)ELSEEX%=EX%+1:OS=EX%
2110 GETFI%,1:LSETO3$=MKI$(EX%):LSETO5$=MKI$(OA):PUTFI%,1:ER%=0:RETURN
2199 '
                FREE OLD SECTOR (OS, RETURNS: OE, OL, ER%)

2200 GETFI%,OS:OE=CVI(OE$):OL=CVI(OL$):LSETOM$="F":LSETOL$=MKI$(OA):OA=OS:
       PUTFI%,OS:GOTO62110
```

SECTION VII

MIKSAM

PEACHTREE SOFTWARE™

MIKSAM

Multi Indexed Keyed Sequential Access Method

9/25/78

PEACHTREE SOFTWARE[tm]
MIKSAM

Multi Indexed Keyed Sequential Access Method

MIKSAM is a general purpose file access program written in Assembly Language to be used as a co-routine with Altair or Microsoft BASIC. MIKSAM is called from BASIC whenever MIKSAM file access is to be performed; BASIC is called from MIKSAM whenever BASIC file access is to be performed.

MIKSAM requires about 7.5K of RAM, which includes about 1K for file buffers.

MIKSAM is distinguished by nine characteristics:

1) It allows for variable length keys (up to 26 bytes each).
2) Records are of fixed length (up to 251 bytes each).
3) There may be up to 255 orders of keys.
4) There may be up to 255 keys per record.
5) There may be multiple records per key (limited by file space).
6) Records may be accessed randomly from any of its keys.
7) Records may be accessed randomly via its record pointers.
8) Records may be sequentially accessed.
9) Files are dynamically self-policing; no extra update or optimization programs are required.

There are ten BASIC commands that allow the user access to MIKSAM:

BASIC Line	Command
60000	SETUP
60100	CREATE FILE
61000	OPEN FILE
61100	CLOSE FILE
61200	SEEK KEY
61300	CREATE KEY RECORD
61400	CREATE KEY
61500	DELETE KEY
61600	GET RECORD
61700	PUT RECORD
61800	FILE STATUS

MIKSAM QUICK REFERENCE

Line	
60000	SETUP
60100	CREATE FILE (NA$, DA$, FI%, DR%, FL%, NO$, ER%)
61000	OPEN FILE (NA$, FI%, DR%, RA%(FI%), KE$(FI%), PT!(FI%), ER%)
61100	CLOSE FILE (FI%, ER%)
61200	SEEK KEY (FI%, RA%(FI%), KE$(FI%),PT!(FI%), MO%, ER%)
61300	CREATE KEY RECORD (FI%, RA%(FI%), KE$(FI%), PT!(FI%), RE$(FI%), ER%)
61400	CREATE KEY (FI%, RA%(FI%), KE$(FI%), PT!(FI%), ER%)
61500	DELETE KEY (FI%, RA%(FI%), KE$(FI%), PT!(FI%), ER%)
61600	GET RECORD (FI%, PT!(FI%), RE$(FI%), ER%)
61700	PUT RECORD (FI%, PT!(FI%), RE$(FI%), ER%)
61800	FILE STATUS (FI%, RS$, ER%)

SEEK KEY MODES

MODE	DESCRIPTION
0	Rewind to first of RA% keys
1	Seek key on RA% and KE$
2	Seek next key
3	Seek next set
4	Seek Key Pointer on RA%, KE$ and PT!

MIKSAM TERMINOLOGY

Page - A 256 byte area in logical file space comprised of two BASIC records.

Offset - A byte value in the range $\emptyset \leq$ Offset ≤ 255.

Key - An area of variable length from two to twenty-eight bytes long. The first byte of this area is the length of the area (including itself in the length count). The second byte of this area is the rank number (RA%) of the Key (1=primary, 2=secondary, etc.) up to 255 Ranks. The remaining bytes comprise the key value (KE$) as specified by the user.

Record Pointer - A three byte (Page, Offset) pair which refers to a Record in the file.

Node Pointer - A three byte field containing a BASIC Record number and $\emptyset$, used to reference a Node or Leaf.

Record - An area containing data indexed by one or more Keys. The first three bytes of the Record's data area is reserved for future use. The fourth byte of the Record's data area is a reference count used to indicate how many Keys refer to this Record. ($\emptyset \leq$ reference count ≤ 255. A reference count of $\emptyset$ implies the Record is deleted. Upon creation of a file the Record's length is specified. This length includes the four bytes of overhead mentioned above, leaving length-4 bytes for the user. These four bytes are automatically skipped in the GET REC and PUT REC subroutines. Within a BASIC program the user can assume the Record length is four less than specified at file creation time.

Node - A 128 byte area containing a one byte length followed by alternating Node Pointers and Keys followed by a final Node Pointer. Node Pointers refer to Node or Leafs containing Keys lexigraphically less than or equal to the right-adjoining Key; the final Node Pointer refers to a Node or Leaf containing Keys which are lexigraphically greater than the last Key.

Leaf - A 128 byte area similar to a Node except that all pointers refer to the Record associated with the right-adjoining Key. The final Pointer refers to the Record associated with the Leaf's parent Key. The last pointer in the file is a null pointer.

Tree Structure - MIKSAM data structure. The tree structure is comprised of the "Root" Node or Leaf. Each pointer from a Node refers to a lower level Node or Leaf. All Leafs are on the same level of the tree (equal to TREEHEIGHT) and pointers from the Leaf refer to Records.

For further reference as to how the actual keys and records are manipulated, see The Art of Computer Science, Volume III Sorting and Searching, p471-48 by E.D. Knuth and Organization & Maintenance of Larger Ordered Indexes, Acta Informatica 1, 173-189 (1972) by R. Bayer & E. McCreight.

MULTI-INDEXED KEYED SEQUENTIAL ACCESS METHOD

(MIKSAM)

CHARACTERISTICS

1. Variable length keys (up to 26 bytes each).
2. Fixed length records (up to 252 bytes each).
3. Multiple-orders of keys (primary, secondary, etc., up to 255 order levels).
4. Multiple keys per record (up to 255).
5. Multiple records per key (limited by file space only).
6. Random access to a record via any of its keys.
7. Random access to a record via its record pointer.
8. Sequential access to a record.

COMMAND SUMMARY

Line

```
60000  SETUP
60100  CREATE FILE (NA$, DA$, FI%, DR%, FL%, NO$, ER%)
61000  OPEN FILE (NA$, FI%, DR%, RA%(FI%), KE$(FI%),
       PT!(FI%), ER%)
61100  CLOSE FILE (FI%, ER%)
61200  SEEK KEY (FI%, RA%(FI%), KE$(FI%),PT!(FI%), MO%, ER%)
61300  CREATE KEY RECORD (FI%, RA%(FI%), KE$(FI%), PT!(FI%),
       RE$(FI%), ER%)
61400  CREATE KEY (FI%, RA%(FI%), KE$(FI%), PT!(FI%), ER%)
61500  DELETE KEY (FI%, RA%(FI%), KE$(FI%), PT!(FI%), ER%)
61600  GET RECORD (FI%, PT!(FI%), RE$(FI%), ER%)
61700  PUT RECORD (FI%, PT!(FI%), RE$(FI%), ER%)
61800  FILE STATUS (FI%, RS$, ER%)
```

DESCRIPTION OF USER FUNCTIONS

60000 - SETUP

This routine initializes the variables, allocates space for the MIKSAM variables, and sets up the MIKSAM buffers. No parameters are needed.

60100 - CREATE FILE (NA$, DA$, FI%, DR%, RL%, NO$, ER%)

Create a MIKSAM file on the disk. The file is not opened after creation and must be explicitly opened before use. There are seven parameters.

NA$ - the name of the MIKSAM file to be created, up to eight bytes long.
DA$ - an 8 character string representing the date in the form MM/DD/YY.
FI% - a temporary file number.
DR% - the disk drive the file is to be created on
RL% - the fixed record length for this file, including a 4 byte overhead for each record
NO$ - a string of up to 64 characters saved with the file for user use only (e.g., user name, version, system name, etc.)
ER% - an error number having the value:

0 - no error
111 - a file with name NA$ aleadt exists on drive DR%.

61000 - OPEN FILE (NA$, FI%, DR%, RA%(FI%), KE$(FI%), PT!(FI%), ER%)

Open a MIKSAM file on the disk. Initialize header information and buffer. If ER%=Ø the key index is set to the first rank and key in the file. The key index is undefined otherwise. There are seven parameters:

NA$ - the name of the MIKSAM file to be opened.
FI% - the file number for this file while opened.
DR% - the disk drive the file resides on.
RA%(FI%) - the rank of the key to be created.
KE$(FI%) - the key of the record to be created.
PT!(FI%) - pointer value to created record.
ER% - an error number having the value:

0 - no error
112 - MIKSAM file with name, NA$ does not exist on drive, DR%
103 - file is empty.

61100 - CLOSE FILE (FI%, ER%)

Close the opne file FI% and clear out any buffers associated with this file. The header record is saved, There are two parameters:

FI% - the file number of the file to be closed
ER% - an error number having the value:

0 - no error
110 - no file FI% is currently opened.

61200 - SEEK KEY (FI%, RA%(FI%), KE$(FI%), PT!(FI%), MO%, ER%)

Set the key index to a new value determined by the RA% KE$, MO%, parameter. There are six parameters:

FI% - the file number of the file to be sought
RA%(FI%) - depending on MO% this parameter specifies the rank of the key to be found
KE$(FI%) - depending on MO%, this parameter specifies the key of the record to be found
PT!(FI%) - the pointer of the key found or (if MO%=4) the pointer of the key to be found
MO% - set the key index according to:

If MO%=Ø then rewind to the first of the rank RA% keys. KE$ is set to the name of this key.
If MO%=1 then set the key index to the order specified by RA% and the key specified by KE$. (SEEK KEY)
If MO%=2 then set the key index to the next key in the file. RA% and KE$ are set to this rank and key. (SEEK NEXT KEY)
If MO%=3 then set the key index to the next record set, i.e., the next key different from the current key. (Used when there are multiple records per key in the file.) RA% and KE% are set to this order and key. (SEEK NEXT SET)
If MO%=4 then set key index to RA%, KE$, PT!. (SEEK KEY POINTER)

ER% - an error number having the value:

0 - no error.
101 - end of Set
102 - end of Rank reached
103 - limit of File reached
105 - MODE%=1 and there is no rank=RANK% with key=KEY$ in the file

61300 - CREATE KEY RECORD (FI%, RA%(FI%), KE$(FI%), PT!(FI%), RE$, ER%

Create a new record having the value RE$ with rank RA% and key KE$. The key index is set to the key of the record created. The key index is undefined if ER%≠0. There are six parameters:

FI%	- the file number of the file in which the record is to created.
RA%(FI%)	- the rank of the key to be created.
KE$(FI%)	- the key of the record to be created.
PT!(FI%)	- pointer value to created record.
RE$	- a string of length RL%-4 (Record Length) containing the value of the record
ER%	- an error number having the value:

0 - no error
120 - invalid key (parameter)
121 - no room in file

61400 - CREATE KEY POINTER (FI%, RA%(FI%), KE$(FI%), PT!(FI%), ER%)

Create a new key for the current record and leave the pointer at the created key. The current record is pointed to from the key index set by a previous command. The key index is set to the key created is ER%=0. The key index is undefined if ER%≠0. There are five parameters:

FI%	- the file number of the file in which the key is to be CREATEd.
RA%(FI%)	- the order of the key to be created
KE$(FI%)	- the key of the record to be created
PT!(FI%)	- pointer value to created record
ER%	- an error number having the value

0 - no error
115 - Key Index not set
120 - invalid Key or parameter
121 - no room in file

61500 - DELETE KEY POINTER (FI%, RA%(FI%), KE$(FI%), PT!(FI%), ER%)

Delete the key pointer to form the current key index from the file. If this is the last key referring to a record, the record is also deleted. The key index is set to the next key after the deleted key. There are five parameters:

FI% - the file number of the file in which the key is to be deleted
RA%(FI%) - the order of the key to be deleted
KE$(FI%) - the key of the record to be deleted
PT!(FI%) - the pointer value of the record to be deleted
ER% - an error number having the value:

0 - no error
103 - end of file (last key in file deleted)
115 - Key Index not set

61600 - GET RECORD (FI%, PT!(FI%), RE$(FI%), ER%)

Using the key index set by the SEEK command, get the corresponding record and assign it to the string RE$. The length of RE$ must be RL%-4 (the fixed record length for the file). The key index is unmodified by this command. There are four parameters:

FI% - the file number of the file from which the record is to be obtained
PT!(FI%) - the pointer value to created record
RE$(FI%) - a string which will contain the record value obtained from the file
ER% - an error number having the value:

0 - no error
115 - Key Index has not been set

61700 - PUT RECORD (FI%, PT!(FI%), RE$(FI%), ER%)

Using the key index, put into the corresponding record the value of RE$. The length of RE$ must be RL%-4. The key index is unmodified by this command. There are three parameters:

FI% - the file number of the file into which the record is to be placed
PT!(FI%) - the pointer value to created record
RE$(FI%) - a string which contains the record value placed into the file.
ER% - an error number having the value:

0 - no error
115 - Key Index has not been set

61800 - FILE STATUS (FI%, FS$, ER%)

Obtain the status of the file with file number FI% as of the last time the file was closed. Place the information in the string FS$. The file must be opened to use this command. There are three parameters:

FI% - the file number of the file whose status is to be obtained

FS$ - a string containing the header information with the following value:

BYTE	
0-5	"MIKSAM"
6-7	Root of Tree
8-9	Extent of file
10-11	Record Length
12-13	Tree Height
14-15	First available node
16-17	First available record
18-19	First available record offset
20-27	Creation data (MM/DD/YY)
28-35	MIKSAM version date
36-63	Undefined
64-128	NO$ - user defined area

ER% - an error number having the value:

0 - no error

110 - no file FI% is currently opened

RESERVED VARIABLE NAMES

DA$	Date in the form MM/DD/YY	(8 alpha characters)
DR%	Drive the file is mounted on	(integer)
ER%	Error number	(integer)
FI%	File number	(integer)
FS$	File status when file was last closed	(128 byte string)
KE$(FI%)	Key value for file FI%	(0-30 byte string)
MO%	Mode for seek command	(0-4 integer)
NA$	Name of file	(0-8 byte string)
NO$	Note (optionally user defined)	(64 byte string)
PT!(FI%)	Pointer value to record in file FI%	(4 byte number)
RA%(FI%)	Rank of key for file FI%	(0-255 integer)
RE$(FI%)	Record value for file FI%	(0-251 byte string)
RL%	Record length (includes 4 bytes overhead)	(4-255 integer)

ERROR NUMBER MEANINGS

0 - No error
1-99 - BASIC defined errors
101 - End of record set reached
102 - End of rank reached
103 - End of file reached
105 - In the SEEK KEY commands; MODE=1 and there is no rank=RA%, key=KE$ in the file or MODE=4 and there is no rank=RA%, key=KE$, pointer=PTR! in the file.
110 - No file FI% is currently opened
111 - A file with name NA$ already exists on drive DR%
112 - File NA$ is not a MIKSAM file or does not exist
115 - Key index not set
120 - Invalid parameter
121 - File length exceeds disk capacity

LIST OF VARIABLES FOR MIKSAM

DA$	-60140
ER%	-60000, 60110, 60120, 61010, 61020, 61030, 61040, 61110, -61300, 61400, 62010
FI%	-60000, 60130, 60150, 61030, 61040, 61050, 61100, -61300, 61400, 61800, 62010, 63020, 63030, 63040
FS$	-61800
KE$(	-60000, 61300, 61400, 62010, 63040
MO%	-60000, 61050, 62010
NA$	-60100, 60130, 61000, 61030
NO$	-60140
O$(	-60000, 61030, 61040, 61800
O(	-60000, 61040, 61050, 61100, 62010, 63000, 63020, 63030, -63040
Oø$	-60130, 60140, 60150, 61030, 61040, 61100
O1$	-60130, 60140, 60150
O2$	-60130, 60140
O3$	-60130, 60140
O4$	-60130, 60140
O5$	-60130, 60140
O6$	-60130, 60140
O7$	-60130, 60140
O8$	-60130, 60140
O9$	-60130, 60140
OB	-60000, 60010, 61040, 62000, 63000
OI	-60010, 61040, 62000
OJ	-60010, 61040, 62000

LIST OF VARIABLES FOR MIKSAM

OT	-61040, 61050, 61100, 61200, 61300, 61400, 61500, 61600, -61700, 62010, 63000
OT$	-60130, 61030, 61100
PT!(	-60000, 62010
RA%(	-60000, 61050, 62010
RE$(	-60000, 61050, 62010, 63040
RL%	-60140

```
'999 '2/21/79
                MULTI-INDEXED KEYED SEQUENTIAL ACCESS METHOD - MIKSAM

1000 DEFINT O: OB=&H9C50: DEFUSR7=OB:
     DIM O$(5),RA%(5),KE$(5),PT!(5),RE$(5),O(9): FI%=1: ER%=0: MO%=0
1010 FOR OI=&H8C TO &H93: POKE OI+OB,255: NEXT:
     FOR OI=&H94 TO &HA3: POKE OI+OB,0: NEXT:
     OJ=1: FOR OI=&H83 TO &H8A: POKE OI+OB,OJ: OJ=OJ+1: NEXT: POKE OI+OB,0:
     RETURN
1090 '
     CREATE FILE (NA$,DA$,FI%,RL%,NO$,ER%)
1100 ON ERROR GOTO 60110: NAME NA$ AS NA$
 110 ER%=ERR: RESUME 60120
1120 ON ERROR GOTO 54500: IF ER%=58 THEN ER%=111: RETURN
     ELSE IF ER%<>53 THEN ERROR ER% ELSE ER%=0
1130 OPEN"R",FI%,NA$: FIELD FI%,6 AS 00$,2 AS 01$,2 AS 02$,2 AS 03$,2 AS 04$,
     2 AS 05$,4 AS 06$,8 AS 07$,8 AS 09$,28 AS OT$,64 AS 08$
1140 LSET 00$="MIKSAM": LSET 01$=MKI$(2): LSET 02$=01$: LSET 03$=MKI$(RL%):
     LSET 04$=MKI$(0): LSET 05$=04$: LSET 06$=04$+04$: LSET 07$=DA$:
     LSET 08$=NO$: LSET 09$="011679"
1150 PUT FI%,1: FIELD FI%,1 AS 00$,3 AS 01$: LSET 00$=CHR$(4):
     LSET 01$=MKI$(0)+CHR$(0): PUT FI%,2: CLOSE FI%: RETURN
     OPEN FILE (NA$,FI%,RA%,KE$,PT!,ER%)
2000 ON ERROR GOTO 61010: NAME NA$ AS NA$
2010 ER%=ERR: RESUME 61020
2020 ON ERROR GOTO 54500: IF ER%=53 THEN ER%=112: RETURN
     ELSE IF ER%<>58 THEN ERROR ER% ELSE ER%=0
2030 OPEN "R",FI%,NA$: FIELD FI%,128 AS O$(FI%): FIELD FI%,6 AS OT$,14 AS 00$:
     GET FI%,1: IF OT$<>"MIKSAM" THEN ER%=113: CLOSE FI%: RETURN
2040 O(0)=FI%*256:FOROT=1TO7:O(OT)=CVI(MID$(00$,OT*2-1,2)):NEXT:
     O(8)=VARPTR(O$(FI%)):O(9)=VARPTR(ER%):OI=&H32A+OB:OJ=&H348+OB:
     POKEOI,OJAND255:POKEOI+1,((OJ/256)AND255):OI=&H350+OB:OJ=&H352+OB:
     POKEOI,OJAND255:POKEOI+1,((OJ/256)AND255):GOSUB63000
2050 RE$(FI%)=SPACE$(O(3)): RA%(FI%)=0: MO%=0: OT=2: GOTO 62000
2090 '
     CLOSE FILE (FI%,ER%)
2100 ON ERROR GOTO 61110: FIELD FI%,6 AS OT$,14 AS 00$: ON ERROR GOTO 54500:
     OT=1: GOSUB 62000: GET FI%,1: OT$="": FOR OT=1 TO 7:
     OT$=OT$+MKI$(O(OT)): NEXT: LSET 00$=OT$: PUT FI%,1: CLOSE FI%: RETURN
2110 ER%=110: GOTO 63010
```

```
61190 '
        SEEK KEY (FI%,RA%,KE$,PT!,MO%,ER%)
61200 OT=2: GOTO 62000
61290 '
        CREATE KEY RECORD (FI%,RA%,KE$,PT!,RE$,ER%)
61300 IF LEN(KE$(FI%))>26 THEN ER%=120: RETURN ELSE OT=3: GOTO 62000
61390 '
        CREATE KEY POINTER (FI%,RA%,KE$,PT!,ER%)
61400 IF LEN(KE$(FI%))>26 THEN ER%=120: RETURN ELSE OT=4: GOTO 62000
61490 '
        DELETE KEY POINTER (FI%,RA%,KE$,PT!,ER%)
61500 OT=5: GOTO 62000
61590 '
        GET RECORD (FI%,PT!,RE$,ER%)
61600 OT=6: GOTO 62000
61690 '
        PUT RECORD (FI%,PT!,RE$,ER%)
61700 OT=7: GOTO 62000
61790 '
        FILE STATUS (FI%,FS$,ER%)
61800 GET FI%,1: FS$=O$(FI%): RETURN
61990 '
        CALL MIKSAM
   0 OI=&H32A+OB:OJ=&H348+OB:POKEOI,OJAND255:POKEOI+1,((OJ/256)AND255):
        OI=&H350+OB:OJ=&H352+OB:POKEOI,OJAND255:POKEOI+1,((OJ/256)AND255)
62010 O(0)=OT+FI%*256:O(1)=VARPTR(RA%(FI%)):O(2)=VARPTR(KE$(FI%)):
        O(3)=VARPTR(PT!(FI%)):O(4)=VARPTR(RE$(FI%)):O(5)=VARPTR(ER%):O(6)=MO%
63000 OT=USR7(VARPTR(O(0))):
        ON PEEK(OB+&H11)+1 GOTO 63010,63020,63030,63040,63050
63010 RETURN
63020 GET FI%,O(0): GOTO 63000
63030 PUT FI%,O(0): GOTO 63000
63040 KE$(FI%)=SPACE$(O(0)):
        O(2)=VARPTR(KE$(FI%)): O(4)=VARPTR(RE$(FI%)): GOTO 63000
63050 GOTO 63000 '       O(1)= <FREE SPACE ON DISK>: GOTO 63000
```