HP 9800 Computer Systems

Data Transfer Utility

For the HP-85 to 9835 and 9845

HEWLETT PACKARD

Computer Museum

Data Transfer Utility

Part No. 09835-10051

Hewlett-Packard Desktop Computer Division
3404 East Harmony Road, Fort Collins, Colorado 80525

Copyright by Hewlett-Packard Company 1979

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Printing History

New revisions of this manual will incorporate all material updated since the previous revision.

The manual printing date and part number indicate its current revision. The printing date changes when a new revision is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

January 1980...Revision A

November 1980...Revision B. Updated pages: i, 2, 3, 13, 15

Important

The tape cartridge or disc containing the programs is very reliable, but being a mechanical device, is subject to wear over a period of time. To avoid having to purchase a replacement medium, we recommend that you immediately duplicate the contents of the tape onto a permanent backup tape or disc. You should also keep backup copies of your important programs and data on a separate medium to minimize the risk of permanent loss.

Table of Contents

Introduction

This set of utilities provides a method by which the 9835 or 9845 can read data from tapes generated by the HP-85. The HP-85 uses a form of SIF (Standard Interchange Format) to write data on its tapes. Therefore the SIF binaries on the 9835/9845 can be used to read the tapes; however, since the HP-85's internal storage format is different from the 9835/9845, it is necessary to take the information on the HP-85's tapes and decode it so as to be useable on the 9835/9845. This decoding process is what is provided by the HP-85 to 9835/9845 Data Utilities.

System Configuration

9845A Desktop Computer
or
9845B Desktop Computer
or
9845C Desktop Computer
or
9835A or B Desktop Computer

Standard Memory is all that is needed; however, larger memory may be needed for larger data files.

Ordering Information

Complete Pack	09835-10050
Manual	09835-10051
Cartridge	09835-10054

Explanation of Files

The following files can be found on the tape cartridge supplied with this utility library:

AUTOST —	This 9845B/C program simply brings "HELP" into memory and runs it.
HELP —	This program lists out instructions on how to load and run the program which uses the utilities to print out the entire contents of a data file, record by record.
45ASIF —	This is the SIF binary which is used by the 9845A to read the bit patterns from an HP-85 tape cartridge.
45BSIF —	This is the SIF binary which is used by the 9845B or C to read the bit patterns from an HP-85 tape cartridge.
35ASIF —	This is the SIF binary which is used by the 9835A or B to read the bit patterns from an HP-85 tape cartridge.
DUMPUT —	This file contains the utilities which are used to read the information stored on an HP-85 tape. In order to conserve memory, there are no remarks or comments imbedded in the programs which explain what each particular section of code is doing.
REMARK —	This file contains the same thing as DUMPUT (see above), except that complete annotations are also provided.
PRTUT —	This program is a front-end application program which uses the Data Utilities. It can be used to print the entire contents of a data file, record by record, or it can be used as an example on how to use the provided utilities.
FACTO —	This program is also a front-end application program which uses the Data Utilities. It is used as an example and is explained fully in Appendix A.
HSTDAT —	This is a data file used by "FACTO."
STRIP —	This program is a comment stripper. It is used to strip comments from a program that is SAVE'd on a DATA file so that it doesn't take as much room when it is brought into memory. The original program is kept intact, while the stripped program is put on a new data file.
REVID —	This file contains information specifying the revision of the software.

NOTE

The AUTOST file will generate Error 58 (improper file type) on a 9835 or 9845A, as autostart files are machine dependent. This error can simply be ignored. To use the HELP file, type GET "HELP" [EXECUTE].

Explanation of Each Utility and Rules for Use

In general, a set of utilities does not provide a solution to a problem; rather they are used as a tool in solving the problem. The particular problem that needs to be solved will vary from user to user. Each user may need to write a unique application program to solve his own problem, but often several users can use some of the same tools. Here, one user may want to scan an HP-85 file serially and graph successive coordinate pairs, while another user may want to copy a data base to a hard disk on a record-by-record basis. Obviously, each user's application program will differ radically. Still, each will be able to make use of the HP-85 to 9835/9845 Data Utilities. First, though, it is necessary to understand how to use them.

1. Certain information is required by all the utilities. This information is accessible through the COM statement. The user must provide the following COM statement in his main program:

   COM Index$[10],Buffer$[512],INTEGER Pointer, Recno,Norecs,Lrecl,Nologs,Lrec-no,File,Bytes

   The COM statement is included in the DUMPUT file as the first line for the user's convenience.
2. Since these programs cannot access the HP-85's tape directory, the user must have a copy of a CATalogue from the HP-85 in front of him in order to provide information regarding the file's number, logical record length, and number of logical records.
3. The COM values listed below must be defined prior to using any of the rest of the utilities. The utility SUB Define_live will allow the user to define the values g, d, and e from the keyboard, while the value for c is computed. SUB Define_live also calls SUB Record, which defines values a, b, f, and h. Index$ is defined automatically by the function FNType, while Buffer$ is defined by SUB Record, and maintained by the rest of the utilities. For non-interactive definition of the parameters, SUB Define_parm accomplishes the same function by passing the cited values through the parameter list.

   a. Pointer — the next available byte within the current physical record.
   b. Recno — the current physical record
   c. Norecs — the number of physical records in the file
   d. Lrecl — logical record length
   e. Nologs — number of logical records in the file
   f. Lrecno — the current logical record
   g. File — the file number to be read
   h. Bytes — the number of bytes remaining in the current logical record

Computer Museum

4. Here is a list of the utilities available to the user, along with an explanation of what each one does:

   a. SUB Define_live
   This routine causes the values in the COMmon area to be defined interactively, by having the user enter appropriate values from the keyboard.

b. SUB Define_parm(Tape,File,Lrecl,Nologs)

This routine sets up the common area using the values passed in through the parameter list.

Tape	—	(integer) The select code of the tape drive holding the HP-85 data tape (14 or 15 for a 9845, 15 for a 9835)
File	—	(integer) The number of the file to be read (>0)
Lrecl	—	(integer) The logical record size (4 <=Lrecl <= 32767)
Nologs	—	(integer) The number of logical records in the file (>0)

c. SUB Record (Record)

This subprogram will set the file pointer to the specified record number for random accesses.

Record	—	(full precision) specifies which logical record is to be accessed in following operations

d. DEF FNType

This function provides information regarding the type of the next data item. Here are the values returned:

1	—	full precision numeric
2	—	entire string
3	—	end of file
4	—	end of record
8	—	first part of a string
9	—	middle part of a string
10	—	last part of a string

All other values are unused.

NOTE

The only way to get past an EOF or EOR is to use the SUB Record subprogram to advance the file pointer to the beginning of the next logical record.

e. SUB Get_num(A,Err)

This subprogram will go to the current file position and return a full precision number:

A	—	(full precision) This can be any full precision variable. Overflows and underflows will be handled by DEFAULT ON.
Err	—	(full precision) If nothing goes wrong, Err will be set to zero. If a non-numeric item is found, Err will be set to the type of the next data item (see definitions of variables types under the FNType function).

f. SUB Get_array1(A(*),N,Err)

This subprogram will go to the current file position and return a one-dimensional array of full precision numbers. Here are the parameters:

A(*)	—	(full precision) This can be any legal one dimensional array which has a lower bound of 0 or 1 (the only legal lower bounds on the HP-85), and an upper bound of at least N (see below).
N	—	(full precision) Tells the upper bound of the array segment to be filled
Err	—	(full precision) If nothing goes wrong, Err will be set to zero. If a non-numeric item is found prior to filling the array, Err will be set to the type of the next item, and N will be set to the number of array elements successfully copied. If an error is found in the array, a negative error code will be returned: – 1 — array is not one-dimensional – 2 — array does not contain the subscripts 0 or 1 – 3 — array does not contain the subscript N

NOTE

Because of the way the HP-85 is defined, arrays can only have 0 or 1 as a lower subscript. 0 is sought first, and used if possible; otherwise 1 is used as a lower subscript.

---

g. SUB Get_array2(A(*),N,M,Err)

This subroutine will go to the current file position and return a two-dimensional array of full precision numbers. Here are the parameters:

A(*)	—	(full precision) This can be any legal two dimensional array which has a lower bound of 0 or 1 (the only legal lower bounds on the HP-85), an upper row bound of not less than N, and an upper column bound of not less than M (see below).
N	—	The minimum upper row bound of the array
M	—	The minimum upper column bound of the array
Err	—	(full precision) If nothing goes wrong, Err will be set to zero. If a non-numeric item is found prior to filling the array, Err will be set to the type of the next item, and N and M will be set to the last array element which was successfully copied. If an error is found in the array, a negative error code will be returned: – 1 — array is not two-dimensional – 2 — array does not contain elements (0,0) or (1,1) – 3 — array does not contain the element (N,M)

NOTE

Because of the way the HP-85 is defined, both dimensions are assumed to have the same lower bound. If (0,0) exists, it is used as the lower bound; otherwise (1,1) is used if it exists.

---

h. SUB Get_t_string(A$,Err)

This subprogram will go the current file position and return an entire string, regardless of whether or not it spans several logical (or physical) records.

A$	— A user-dimensioned string
Err	— (full precision) If nothing goes wrong, Err will be set to zero. If a numeric item is encountered, Err will be set to one. If the entire string on the tape will not fit in A$, Err will be set to 2, and A$ will be filled up as far as possible.

i. SUB Get_p_string(A$,Err)

This subprogram will go the current file position and return that part of the string which is contained in the current logical record.

A$	— A user-dimensioned string
Err	— (full precision) If nothing goes wrong, Err will be set to zero. If a numeric item is encountered, Err will be set to one. If the entire string part on the logical record will not fit in A$, Err will be set to 2, and A$ will be filled up as far as possible.

5. In addition to the above list of utilities, there are two more subprograms which are not meant to be used by the user. These subprograms are used for keeping track of the current location within the file and within the current physical record. They are used by several of the other utilities. They are called SUB Newrec and SUB Update.

6. The user may not find it necessary to use all of the above utilities for his particular application. In this case, it is possible to select only those utilities that he needs for inclusion in his program. However, the user should be aware that some of the utilities call other utilities. Thus, the user may need to include a certain utility, even though he doesn't explicitly call it himself. Following is a list of the subprograms that require other utilities to execute, as well as which utilities are required:

```
SUB Define_live
      requires SUB Record
SUB Define_parm
      requires SUB Record
SUB Get_num
      requires DEF FNType
      requires SUB Update
      requires SUB Newrec
SUB Get_array1
      requires SUB Get_num
      requires (refer to the list under SUB Get_num)
SUB Get_array2
      requires SUB Get_num
      requires (refer to the list under SUB Get_num)
```

```
SUB  Get_p_string
       requires DEF FNType
       requires SUB Update
       requires SUB Newrec
SUB  Get_t_string
       requires DEF FNType
       requires SUB Update
       requires SUB Newrec
       requires SUB Get_p_string
DEF  FNType
       stand-alone
SUB  Newrec
       stand-alone
SUB  Update
       stand-alone
SUB  Record
       stand-alone
```

Getting Started

In general, here are the steps a user will want to go through to get the data off the HP-85 tape:

1. Get a listing of the directory of the tape to be dumped (perform the CAT command on the HP-85). This is so you'll be able to supply the Data Utilities with the file's number, its logical record size, and the number of logical records in the file. Here is an example:

logical record size

number of logical records

file number

```
NAME     TYPE   BYTES    RECS  FILE
TEST1    DATA     256       3    1
TEST2    DATA     256       3    2
TEST3    DATA     128       6    3
TEST4    DATA      20      64    4
TEST5    DATA      80       6    5
TEST6    DATA      16      16    6
DRIVER   PROG     256       1    7
```

File TEST1 is file number 1, it has 3 records, each of which is 256 bytes long. File TEST6 is file number 6, it has 16 records, each of which is 16 bytes long. File DRIVER is not a DATA file, but a PROGram file. It cannot be read with the Data Utilities.

2. The user program should be structured in the following manner:

```
10      COM Index$[10],Buffer$[512],INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrec
no,File,Bytes
20      CALL Define_live   ! Define_parm can also be used here
30      CALL Record(1)     ! Select any starting place in the file
40      ! At this point, everything is set up to begin reading data immediately.
50      ! Assuming you know the order of the data on the tape, you can use
60      ! the utilities Get_num, Get_array1, Get_array2, Get_t_string, or
70      ! Get_p_string in any combination to read simple numerics, one dimen-
80      ! sional arrays, two dimensional arrays, total strings, and partial
90      ! strings, respectively.
1000    ! ******************************************************************
1010    ! After the application program which calls the various Data Utilities
1020    !  and performs some task with the data, you must include the Data
1030    !  Utilities themselves (stored on the provided tape cartridge under
1040    !  the name "DUMPUT").
```

Appendix A (Specific Examples)

The following program is an example of how to use the Data Utilities to find the contents of a tape cartridge if you do not already know the order in which the data items are stored. It uses the function FNType to find out what the next data item is in the current logical record. This tells the program which routine to call (Get_num or Get_p_string) to get the data from the HP-85 tape.

This program is provided with the HP-85 to 9835/9845 Data Utilities Library. It is stored on the tape under the file name "PRTUT." To use this program, perform the following steps:

Type: SCRATCH A [EXECUTE]
If you are using a 9845A, Type: LOAD BIN "45ASIF" [EXECUTE]
If you are using a 9845B or C, Type: LOAD BIN "45BSIF" [EXECUTE]
If you are using a 9835 (A or B), Type: LOAD BIN "35ASIF" [EXECUTE]
Type: GET "PRTUT" [EXECUTE]
Type: GET "DUMPUT",1000 [EXECUTE]
Press: RUN

NOTE
On a 9835, two IMPROPER STATEMENT messages will be generated, as the INTERCHANGE IS statement is not recognized. The program, however, will work as expected.

— Sample program listing —

```
1    ! COM Index$[10],Buffer$[512],INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrec
no,File,Bytes
10     ! This demo program shows the use of the HP-85 data file dump utilities.
20     !  The purpose of the program is to print the entire contents of a user-
30     !  specified data file.  It is necessary for the user to have a listing
40     !  of his HP-85 tape's directory in front of him in order to be able to
50     !  specify 1) the file number, 2) the logical record size, and 3) the
60     !  number of logical records in the file.␊
70     ! The demo program calls the following subset of utilities:
80     !  SUB Define_live
90     !  SUB Record
100    !  DEF FNType
110    !  SUB Get_num
120    !  SUB Get_p_string␊
130    ! This program is not a comprehensive catch-all tape handler.  It is an
140    !  example on how to use the provided utilities in a specific manner.
150    !  If the program does not suit the needs of a user, the user should be
160    !  able to take the example as a guide, and write his own program that
170    !  does suit his particular application.␊
180    DIM A$[1024]
190    CALL Define_live                               ! Allow the user to set up
200    PRINT "File #";File                            !  a file interactively
210    N=0
220    FOR I=1 TO Nologs
230    CALL Record(I)                                 ! Set pointer to beginning
240    PRINT LIN(1),"Record #";I                      !  of each logical record
```

```
250 More:    ! Check type of next data item
260   ON FNType GOTO Num,Str,Eof,Eor,Num,Num,Num,Bstr,Mstr,Lstr
270 Num:     CALL Get_num(A,E)                        ! Number is next item
280          N=N+1
290          IF N>4 THEN GOSUB Lf
300          PRINT USING "#,MD.11DE,X";A
310 Resume:          !
320          IF Bytes=Lrecl THEN Nexti                ! <-- ****** TRICK !
330                                                   ! Bytes tells how many bytes
340                                                   !  are remaining in a logical
350                                                   !  record.  If Bytes=Lrecl,
360                                                   !  then a new logical record
370                                                   !  has been set up because
380                                                   !  the sequential accesses
390                                                   !  have scanned past the end
400                                                   !  of the record without
410                                                   !  finding an EOR mark.
420          GOTO More
430 Lf:      PRINT
440          N=1
450          RETURN
460 Str:     GOSUB Dumpnum                            ! String found.  Flush any
470          PRINT "Total ";                          !  numbers in the print
480          GOTO String                              !  buffer and get the string.
490 Bstr:    GOSUB Dumpnum                            ! Begginning string
500          PRINT "Beginning ";
510          GOTO String
520 Mstr:    GOSUB Dumpnum                            ! Middle string
530          PRINT "Middle ";
540          GOTO String
550 Lstr:    GOSUB Dumpnum                            ! End of string
560          PRINT "End of ";
570 String: !
580          PRINT "String:"
590          CALL Get_p_string(A$,Err)                ! Fetch the string
600          PRINT "Length: ";LEN(A$)                 ! Print the length
610          FOR L=1 TO LEN(A$)-71 STEP 70            ! Print the string in groups
620             PRINT USING 630;A$[L;70]              !  of at most 70 characters.
630             IMAGE "␛Y",70A,"␛Z␍␊  "               !  The escape codes in the
640                                                   !  IMAGE statements will
650                                                   !  cause any control codes
660                                                   !  in the string to be
670                                                   !  printed.  The 'Escape Z'
680                                                   !  at the end of every string
690                                                   !  is supplied by the IMAGE,
700                                                   !  not by the string itself.
710             NEXT L
720          PRINT USING 730;A$[L,LEN(A$)]
730          IMAGE "␛Y",K,"␛Z␍␊  "
740          GOTO Resume
750 Eof:     PRINT "EOF"
760          GOTO Nexti
770 Eor:     PRINT "EOR"
780          GOTO Nexti
790 Nexti:      GOSUB Dumpnum                         ! Next logical record can be
800                                                   !  processed, so dump out any
810                                                   !  numbers left in the print
820                                                   !  buffer.
830          NEXT I
840          PRINT LIN(1),"DONE"
850          STOP
860 Dumpnum:         !
870          IF N<>0 THEN PRINT
880          N=0
890          RETURN
```

Here's another specific example. Suppose the HP-85 is being used as a data collection station for quality control of a bunch of capacitors coming off an assembly line. Capacitors are selected for testing at random, and a test is run on them 32 times to test for values remaining within a certain tolerance of a given mean. The HP-85 will accept or reject components at the test station, but data from several test stations is being taken, and the plant manager may want to be able to see the "big picture," or what the failure rates overall are, and what their characteristics are. To accomplish this, a 9845 is used as a central information station to take the data from all of the HP-85 driven test stations and reduce it to a mean and variance for each test. It will then plot a histogram of all the tests performed every day. So at five o'clock, all of the test technicians bring their tapes to the 9845 and have them read. The 9845 takes the data from ten data tapes, each of which holds the results of 100 tests. Each test data set of 32 numbers is reduced to a mean and variance, which in turn is stored on the 9845's second tape drive. The next morning, the Test Supervisor uses this data to run the histogram program.

The program which performs the reading of the tapes as outlined above is stored on the provided tape cartridge under the file name "FACTO." To use this program, perform the following steps:

Type: SCRATCH A [EXECUTE]
Type: MASS STORAGE IS ":T14"[EXECUTE]
Insert the Data Utilities tape in the left transport of the 9845, and the HP-85 data tape to be read in the right transport.
If you are using a 9845A, Type: LOAD BIN "45ASIF" [EXECUTE]
If you are using a 9845B or C, Type: LOAD BIN "45BSIF" [EXECUTE]
Type: GET "FACTO" [EXECUTE]
Type: GET "DUMPUT",1000 [EXECUTE]
Press: RUN

The program will issue prompts whenever it requires operator intervention.

— Sample program listing —

```
10     OPTION BASE 1
20   ! COM Index$[10],Buffer$[512],INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrec
no,File,Bytes
30     DIM A(32)
40     ASSIGN #1 TO "HSTDAT:T14"
50     FOR I=1 TO 10
60        DISP "Insert tape for test station #";I;"in T15 and press CONT"
70        BEEP
80        PAUSE
90        CALL Define_parm(15,16,256,100)
100       FOR Record=1 TO 100
110       DISP "Do not disturb -- I'm busy reading this tape --";Record;"/100"
120          CALL Record(Record)
130          CALL Get_array1(A(*),32,Err)
140          IF Err THEN Disaster
150          CALL Reduce(A(*),Mean,Variance)
160          PRINT #1;Mean,Variance
170          PRINT Mean;Variance
180       NEXT Record
190    NEXT I
200    DISP "ALL DONE"
210    BEEP
```

```
220   ASSIGN #1 TO *
230   REWIND ":T15"
240   REWIND ":T14"
250   STOP
260 Disaster:    BEEP
270              DISP "FAILURE"
280              PRINTER IS 16
290              PRINT PAGE;"Program failure on tape #";I;", test #";J;"."
300              PRINT "Notify the programming staff immediately."
310              STOP
320   SUB Reduce(A(*),Mean,Var)
330   X1=X2=0
340   N=ROW(A)
350   FOR I=1 TO N
360      X1=X1+A(I)
370      X2=X2+A(I)*A(I)
380   NEXT I
390   Mean=X1/N
400   Var=(X2-X1*X1/N)/(N-1)
410   SUBEND
```

Appendix B (Annotated Listings of Data Utilities)

```
1000  COM Index$[10],Buffer$[512],INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrec
no,File,Bytes

              ********************11/01/79*****************

1020  SUB Define_live                                   ! Define COM parmameters
1030  OVERLAP                                           !  from keyboard
1040  COM Index$,Buffer$,INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrecno,File,B
ytes
1050  Buffer$=""
1060  Pointer=Recno=Norecs=Lrecl=Nologs=Lrecno=File=Bytes=0
1070 Interchange: ! This section is unnecssary for the 9835
1080            INPUT "Which tape drive is the HP-85 tape in (14 or 15)?",Tape
1090            IF (Tape=14) OR (Tape=15) THEN Settape
1100            BEEP
1110            DISP Tape;"is illegal ! "
1120            WAIT 750
1130            GOTO Interchange
1140 Settape: !
1150            INTERCHANGE IS Tape                     ! The 9835 will reject this
1160 Getfile:   File=PI
1170            INPUT "Which file number do you want to dump?",File
1180            IF File<>PI THEN Checkfile
1190            BEEP
1200            DISP "No file specified.  Try again."
1210            WAIT 750
1220            GOTO Getfile
1230 Checkfile: IF (File>0) AND (INT(File)=File) THEN Fileokay
1240            BEEP
1250            DISP "Illegal file number.  Try again."
1260            GOTO 1210
1270 Fileokay:  FIND File
1280 Specs:     INPUT "Record length  (in bytes)?",Lrecl
1290            IF (Lrecl>4) AND (Lrecl<=32767) THEN Repeat
1300            BEEP
1310            DISP "Illegal logical record length"
1320            WAIT 750
1330            GOTO Specs
1340 Repeat:    INPUT "File size (in logical records)?",Nologs
1350            IF Nologs>0 THEN Compute
1360            BEEP
1370            DISP "Illegal number of logical records"
1380            WAIT 750
1390            GOTO Repeat
1400 Compute:   X=Nologs*Lrecl/256                      ! Find number of physical
1410            Norecs=INT(X)                           !  records used by file
1420            IF FRACT(X) THEN Norecs=Norecs+1
1430            CALL Record(1)                          ! Set pointer to default
1440  SUBEND                                            !  (beginning of file)

              *****************************************

1460 Subdefine_parm:!
1470  SUB Define_parm(INTEGER Tape,File,Lrecl,Nologs) ! Define COM area through
1480                                                    !  the parameter list
1490  COM Index$,Buffer$,INTEGER Pointer,Recno,Norecs,Lrecl1,Nologs1,Lrecno,File
1,Bytes
```

Computer Museum

```
1500  Buffer$=""
1510  Pointer=Recno=Norecs=Lrecl1=Nologs1=Lrecno=File1=Bytes=0
1520  OVERLAP
1530  IF (Tape=14) OR (Tape=15) THEN Tapeokay
1540  BEEP
1550  DISP "ILLEGAL TAPE SELECT CODE -- ";Tape
1560  STOP
1570 Tapeokay: !
1580           INTERCHANGE IS Tape                ! The 9835 will reject this
1590           IF File>0 THEN Fileokay
1600           BEEP
1610           DISP "ILLEGAL FILE NUMBER -- ";File
1620           STOP
1630 Fileokay: FIND File
1640           File1=File
1650           IF (Lrecl>4) AND (Lrecl<=32767) THEN Lreclokay
1660           BEEP
1670           DISP "FAULTY LOGICAL RECORD LENGTH --";Lrecl
1680           STOP
1690 Lreclokay: Lrecl1=Lrecl
1700           IF Nologs>0 THEN Nologsokay
1710           BEEP
1720           DISP "ILLEGAL NUMBER OF RECORDS -- ";Nologs
1730           STOP
1740 Nologsokay: Nologs1=Nologs
1750           X=Nologs*Lrecl/256
1760           Norecs=INT(X)                      ! Compute # of physical
1770           IF FRACT(X) THEN Norecs=Norecs+1   !  records used by the file
1780           CALL Record(1)
1790 SUBEND

                ******************************************

1810 Subrecord: !
1820  SUB Record(Record)
1830                                              ! Find a logical record
1840  COM Index$,Buffer$,INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrecno,File,B
ytes
1850  Lrecno=Record                               ! Set COM value
1860  B=(Lrecno-1)*Lrecl+1                        ! Find byte number of logical
1870                                              !  record within the file
1880  Pointer=B MOD 256                           ! Find the byte number of the
1890                                              !  logical record within the
1900                                              !  physical record
1910  R=B DIV 256                                 ! Find the physical record
1920                                              !  where the logical record
1930                                              !  starts
1940  RDELIMITER IS ""
1950  IF NOT LEN(Buffer$) THEN Fresh              ! Check for Buffer$ empty
1960  IF R=Recno THEN Out                         ! Check for Buffer$ already
1970                                              !  having the right contents
1980  IF (R<>Recno+1) AND (R<>Recno-1) THEN Fresh ! Check for one right record
1990  IF R=Recno+1 THEN 2040
2000  Buffer$[257]=Buffer$[1,256]                 ! This code executed if lower
2010  Recno=R                                     !  record is the spare
2020  SREAD R;Buffer$[1,256]
2030  GOTO Out
2040  Buffer$=Buffer$[257]                        ! This branch taken if upper
2050  GOTO Spare                                  !  record is the right one
2060 Fresh:  !
2070  SREAD R;Buffer$[1,256]
```

```
2080 Spare:  !
2090  Recno=R
2100  IF Recno+1<Norecs THEN SREAD Recno+1;Buffer$[257,512]
2110 Out:  Bytes=Lrecl
2120  SUBEND
```

```
               ******************************************
```

```
2140 Deffntype: !
2150  DEF FNType                                   ! Find type of next data item
2160  ! 1 is numeric, 2 is total string, 3 is EOF, 4 is EOR, 8 is beginning of
2170  !  string, 9 is middle string, 10 is end string
2180  COM Index$,Buffer$,INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrecno,File,B
ytes
2190  IF NOT LEN(Index$) THEN GOSUB Define
2200  IF Lrecno>Nologs THEN RETURN 3               ! Eof
2210  X=POS(Index$,Buffer$[Pointer;1])
2220  IF X=0 THEN X=1                              ! Any non-string is a numeric
2230  RETURN X
2240  DATA 15,223,255,239,15,15,15,207,127,111
2250 Define: ! Set up Index$ to be able to interpret header info.
2260  FOR I=1 TO 10
2270  READ X
2280  Index$[I]=CHR$(X)
2290  NEXT I
2300  RETURN
2310  FNEND
```

```
               ******************************************
```

```
2330 Subget_num: !
2340  SUB Get_num(A,Err)
2350                                               ! Get a full precision number
2360  COM Index$,Buffer$,INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrecno,File,B
ytes
2370  DEFAULT ON
2380     Err=FNType
2390     IF Err<>1 THEN SUBEXIT
2400     Err=0
2410     ! exp.          sign (of mantissa)
2420     ! |E2  E3 |E1  S  |  Exponent is 10's complement, sign 0 = +, 9 = -
2430     ! |D11 D12|D9  D10|  12 digit mantissa stored in BCD format
2440     ! |D7  D8 |D5  D6 |
2450     ! |D3  D4 |D1  D2 |
2460     Zero=NUM(Buffer$[Pointer])                ! zeroth byte
2470     One=NUM(Buffer$[Pointer+1])               ! first byte
2480     Expo=INT(One/16)                          ! compute exponent
2490     Expo=Expo*10+INT(Zero/16)
2500     Expo=Expo*10+Zero MOD 16
2510     Sign=1
2520     IF One MOD 16=9 THEN Sign=-1
2530     IF Expo>499 THEN Expo=-(1000-Expo)
2540     A=0                                       ! initialize mantissa
2550     FOR I=7 TO 2 STEP -1                      ! compute mantissa
2560     Num=NUM(Buffer$[Pointer+I])
2570     A=A*10+INT(Num/16)
2580     A=A*10+Num MOD 16
2590     NEXT I
```

```
2600      A=Sign*A*10^(Expo-11)                     ! because of DEFAULT ON, no
2610      Pointer=Pointer+8                         !  overflow or underflow
2620      IF Pointer>256 THEN CALL Newrec           !  errors will happen
2630      IF Pointer>256 THEN Pointer=Pointer MOD 256
2640      CALL Update(8)
2650   SUBEND

                ***********************************************

2670 Subupdate: !
2680   SUB Update(Len)
2690   ! Update the space remaining in the current logical record
2700   COM Index$,Buffer$,INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrecno,File,B
ytes
2710   Bytes=Bytes-Len
2720   IF Bytes>0 THEN SUBEXIT
2730   Bytes=Lrecl
2740   Lrecno=Lrecno+1
2750   SUBEXIT

                ***********************************************

2770 Subget_array1: !
2780   SUB Get_array1(A(*),N,Err)
2790   ! Get a one-d numeric array
2800   ! Err codes:  0  -- okay
2810   !             >0 -- type of the non-numeric item found before the array
2820   !                   was full
2830   !             <0 -- faulty parameter
2840   !                   -1 -- dimensions are improper or inconsistent
2850   !                   -2 -- array does not have subscripts including 0 or 1
2860   !                   -3 -- array does not have a subscript which includes N
2870   COM Index$,Buffer$,INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrecno,File,B
ytes
2880   Err=0
2890   ON ERROR GOTO Err16                          ! Check to insure that array
2900   X=A(0)                                       !  has proper # of dimensions
2910   ON ERROR GOTO Low1                           ! Find out if lower subscript
2920   L=0                                          !  is 0 or 1, (and if it's
2930   X=A(0)                                       !  legal)
2940   ON ERROR GOTO High                           ! Find out if upper subscript
2950   X=A(N)                                       !  is legal
2960   OFF ERROR
2970   FOR I=L TO N                                 ! Loop until array is filled
2980     CALL Get_num(A(I),Err)
2990     IF Err THEN Bomb
3000   NEXT I
3010   SUBEXIT
3020 Bomb: N=I-1
3030        SUBEXIT
3040 Err16:IF ERRN<>16 THEN 2910                    ! Error trapping routines
3050        Err=-1
3060        SUBEXIT
3070 Low1: IF ERRN<>17 THEN Fatal
3080        ON ERROR GOTO Low2
3090        X=A(1)
3100        L=1
3110        GOTO 2940
3120 Low2: IF ERRN<>17 THEN Fatal
3130        Err=-2
3140        SUBEXIT
```

```
3150 Fatal:BEEP
3160       DISP ERRM$
3170       PAUSE
3180       STOP
3190 High: IF ERRN<>17 THEN Fatal
3200       Err=-3
3210       SUBEXIT
3220 SUBEND
```

```
                *****************************************
```

```
3240 Subget_array2: !
3250  SUB Get_array2(A(*),N,M,Err)
3260  ! Get a 2-D numeric array
3270  ! Err codes:  0  -- okay
3280  !              >0 -- type of the non-numeric item found before the array
3290  !                    was full (N and M contain the subscripts of the last
3300  !                    successfully retrieved element
3310  !              <0 -- faulty parameter
3320  !                    -1 -- dimensions are improper or inconsistent
3330  !                    -2 -- array does not have subscripts including 0,0 or
3340  !                          1,1
3350  !                    -3 -- array does not have a subscript which includes
3360  !                          N,M
3370  COM Index$,Buffer$,INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrecno,File,B
ytes
3380  Err=0
3390  ON ERROR GOTO Err16                           ! Find if array has the
3400  X=A(2,2)                                      !  proper # of dimensions
3410  ON ERROR GOTO Low                             ! Find if lower subscript is
3420  L=0                                           !  0 or 1, and if it's legal
3430  X=A(0,0)
3440  ON ERROR GOTO High                            ! Find if upper subscript is
3450  X=A(N,M)                                      !  legal
3460  OFF ERROR
3470  FOR I=L TO N                                  ! Loop until array is filled
3480     FOR J=L TO M
3490        CALL Get_num(A(I,J),Err)
3500        IF Err THEN Bomb                        ! Check for illegal item
3510     NEXT J
3520  NEXT I
3530  SUBEXIT
3540 Bomb: IF J=L THEN J=M+1                        ! Set M and N to reflect the
3550       IF J<>M+1 THEN I=I+1                     !  last successfully re-
3560       M=J-1                                    !  retrieved element
3570       N=I-1
3580       SUBEXIT
3590 Err16:IF ERRN<>16 THEN 3410                    ! Error recovery routines
3600       Err=-1
3610       SUBEXIT
3620 Low:  IF ERRN<>17 THEN Fatal
3630       ON ERROR GOTO Low1
3640       X=A(1,1)
3650       L=1
3660       GOTO 3440
3670 Low1: IF ERRN<>17 THEN Fatal
3680       Err=-2
3690       SUBEXIT
```

```
3700 Fatal:BEEP
3710        DISP ERRM$
3720        PAUSE
3730        STOP
3740 High:  IF ERRN<>17 THEN Fatal
3750        Err=-3
3760        SUBEXIT
3770 SUBEND
```

```
                    ************************************************
```

```
3790   ! String routines
3800      !
3810      !   header     length         characters
3820      ! |11011111| L2  |  L1  |  B1  |  B2  |...etc...|
3830      !
3840      ! Here are the legal string headers:
3850      ! Type 2 (total string)          -- 11011111
3860      ! Type 8 (beginning of string) -- 11001111
3870      ! Type 9 (middle of string)     -- 01111111
3880      ! Type 10 (end of string)       -- 01101111
3890      !
3900      ! The length of the string is stored in binary form, with the MSB
3910      !  at Pointer+2 and the LSB at Pointer +1
3920      !
3930      ! In order for the header for the total string to appear (as opposed
3940      !  to a partial string), the total string size must be less than or
3950      !  equal to the logical record length (i.e. the string will not cross
3960      !  any logical record boundaries (which are computed anyway)).  If a
3970      !  string crosses logical record boundaries, there will be a three
3980      !  byte header at the beginning of each logical record formatted as
3990      !  shown above.  The length fields in this case will reflect the
4000      !  remaining length of the entire string, not the length of the
4010      !  string segment contained in the current logical record.
4020      !
```

```
                    ************************************************
```

```
4040 Subget_p_string: !
4050  SUB Get_p_string(A$,Err)
4060                                                   ! Get partial string
4070  COM Index$,Buffer$,INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrecno,File,B
ytes
4080  DIM String$[Lrecl-3]
4090  Err=0
4100  Type=FNType
4110  ON Type GOTO Err1,Tstrg,Err1,Err1,Err1,Err1,Err1,Pstrg,Pstrg,Tstrg
4120 Err1:      Err=Type
4130            A$=""
4140            SUBEXIT
4150 Tstrg:Len=NUM(Buffer$[Pointer+1])+256*NUM(Buffer$[Pointer+2])
4160 Entry_string:                                     ! Entry point for Pstrg
4170     String$=""
4180 Dragstring: !
4190     Phybytes=256-(Pointer+2)
4200     IF Phybytes<Len THEN Spill                    ! Check for string spanning
4210                                                   !  physical record boundary
4220     String$[LEN(String$)+1]=Buffer$[Pointer+3;Len]
4230     Pointer=Pointer+3+Len
4240     CALL Update(Len+3)                            ! Allow for string header
4250     IF Pointer>256 THEN CALL Newrec
```

```
4260      IF Pointer>256 THEN Pointer=Pointer MOD 256
4270      ON ERROR GOTO Pad
4280      A$=String$
4290      SUBEXIT
4300 Spill: ! If this branch is taken, then a logical record spans a physical
4310         !  record boundary.
4320      IF Phybytes<=0 THEN Softshoe
4330      Bytes=Bytes-Phybytes
4340      String$[LEN(String$)+1]=Buffer$[Pointer+3;Phybytes]
4350      Len=Len-Phybytes
4360      CALL Newrec
4370      Pointer=-2
4380      GOTO Dragstring
4390      !
4400      !
4410 Softshoe: ! If this branch is taken, then a string header has spanned
4420            !  a physical record boundary.
4430      CALL Newrec
4440      Pointer=-2-Phybytes
4450      GOTO Dragstring
4460      !
4470      !
4480 Pstrg:                                         ! Partial string
4490      Len=Bytes-3
4500      GOTO Entry_string
4510      SUBEND
4520 Pad:                                           ! Error recovery
4530      IF ERRN=18 THEN Pad1
4540      BEEP
4550      DISP ERRM$
4560      PAUSE
4570      STOP
4580 Pad1:ON ERROR GOTO Done
4590      Err=2
4600      FOR I=1 TO LEN(String$)
4610      A$[I;1]=String$[I;1]
4620      NEXT I
4630: !
4640  SUBEND
```

```
               ***********************************************
```

```
4660 Subget_t_string: !
4670  SUB Get_t_string(A$,Err)
4680                                                ! Get total string
4690  COM Index$,Buffer$,INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrecno,File,B
ytes
4700  DIM String$[Lrecl-3]
4710  Err=0
4720  A$=String$=""
4730 Loop: !
4740  Type=FNType
4750  ON Type GOTO Err1,Tstrg,Err1,Eor,Err1,Err1,Err1,Mstrg,Mstrg,Tstrg
4760 Err1:    Err=Type
4770          A$=""
4780          SUBEXIT
4790 Eor:     Len=Bytes                             ! Bypass EOR marks and
4800          Pointer=Pointer+Len                   !  go to the next logical
4810          IF Pointer>256 THEN CALL Newrec       !  record to get the string
4820          IF Pointer>256 THEN Pointer=Pointer MOD 256
4830          CALL Update(Len)
4840          GOTO Loop
```

```
4850 Tstrg:   Loop=0                                 ! Total string header or end
4860          GOTO Callprim                          !  string header will show
4870                                                 !  that entire string has
4880                                                 !  been found
4890 Mstrg:   Loop=1
4900 Callprim:CALL Get_p_string(String$,Err)         ! Get string part
4910          L=LEN(A$)
4920          ON ERROR GOTO Glitch
4930          A$[L+1]=String$                        ! Append new part to rest of
4940          OFF ERROR                              !  string
4950          IF Loop THEN Loop                      ! If entire string not satis-
4960          SUBEXIT                                !  fied, go back for more
4970 Glitch:  IF ERRN<>18 THEN Oops                  ! Error trapping routine
4980          ON ERROR GOTO Quit
4990          FOR I=1 TO LEN(String$)                ! Put as much of string as
5000          A$[L+I]=String$[I;1]                   !  possible into return
5010          NEXT I                                 !  variable
5020 Quit:    Err=2
5030          SUBEXIT
5040 Oops:    BEEP
5050          DISP ERRM$
5060          PAUSE
5070          STOP
```

```
                   ************************************************
```

```
5090 Subnewrec:   !
5100  SUB Newrec                                     ! Set up new physical record
5110  COM Index$,Buffer$,INTEGER Pointer,Recno,Norecs,Lrecl,Nologs,Lrecno,File,B
ytes
5120  Recno=Recno+1
5130  Buffer$=Buffer$[257]
5140  IF Recno<Norecs-1 THEN SREAD Recno+1;Buffer$[257,512]
5150  SUBEND
```

Appendix C (Data Formats)

This section explains the way that data is stored on the HP-85's tapes.

All numeric information is stored in full-precision form. A full precision number takes 8 bytes of information. This is sufficient to store the sign of the number, a 10's complement BCD exponent (ranging from + or − 499), and a normalized 12-digit BCD mantissa. Short and integer precision numbers are converted to full precision when they are stored on tape. Arrays are stored on tape simply as a whole bunch of simple numbers. For instance, it is equally valid to read a ten-element array from a file as it is to read ten simple numbers. Two dimensional arrays are stored in row-major order — that is, the entire first row is stored on tape, then the entire second row, etc. For example, if the HP-85 printed this set of numbers on the tape in sequential order:

3,6,4,6,2,3,6,5,9,8,7,4

it would be perfectly acceptable to read them back into a two dimensional array with dimensions of 3x4:

3,6,4,6
2,3,6,5
9,8,7,4

Strings have a special three-byte header which tells the length of the string and the type of string. The string type information is useful when strings cross logical record boundaries. If an entire string is contained in the current logical record, the string is said to be a "total" string. If the string is too long for one logical record, then it can be a "beginning" of a string, the "middle" of a string, or the "end" of a string. The headers indicating the "middle" or "end" of a string will be inserted at the beginning of every record which the string spans. The associated length field tells how long the entire remaining string is, not just the number of characters in the current string segment.

It is worth pointing out here that the HP-85 stores strings differently than the 9835 and 9845. The 9835 and 9845's string headers take up four bytes instead of three. So if the user wants to copy a data file from an HP-85 tape to a 9845 tape, he should be aware that because of the differences in string headers, his destination file will not necessarily be exactly the same as his source file. For example, suppose that the source file (the HP-85 file) has a logical record size of 16 bytes, and contains the string "THIS IS A LONG STRING WITH 68 CHARACTERS WHICH SPANS SEVERAL RECORDS." By using the utility SUB Get_t_string, it is possible to get the entire string from the file and then do a PRINT# to write it on a 9845 tape. However, as the following program illustrates, the string will not be written on the destination file in quite the same manner as it was on the source file.

```
10    ASSIGN #1 TO "FILE" ! Assign the file
20    DIM A$[80]
30    FOR I=1 TO 5
40    READ #1,I             ! Position the pointer at the Ith logical record
50    READ #1;A$            ! Read the string
60    PRINT A$              !  and print it
70    PRINT
80    NEXT I                ! Repeat for five records
90    STOP
```

HP-85 Output:

```
THIS IS A LONG STRING WITH 68 CHARACTERS WHICH SPANS SEVERAL RECORDS

G STRING WITH 68 CHARACTERS WHICH SPANS SEVERAL RECORDS

 68 CHARACTERS WHICH SPANS SEVERAL RECORDS

S WHICH SPANS SEVERAL RECORDS

 SEVERAL RECORDS
```

9845 Output:

```
THIS IS A LONG STRING WITH 68 CHARACTERS WHICH SPANS SEVERAL RECORDS

NG STRING WITH 68 CHARACTERS WHICH SPANS SEVERAL RECORDS

TH 68 CHARACTERS WHICH SPANS SEVERAL RECORDS

TERS WHICH SPANS SEVERAL RECORDS

PANS SEVERAL RECORDS
```

This phenomenon will occur any time a string spans a logical record boundary. It is not a problem if the user is only interested in accessing his data on the destination file in a serial manner. It may, however, cause problems if the application program expects to find a certain segment of a string starting at a certain logical record. It could also cause a file overflow on the destination file if it is created to be exactly the same size as the source file, and the source file is entirely filled up with strings. If the first case is true, then it is recommended to break the long string up into smaller strings which do not cross record boundaries. If the second case is true, change the file and/or record size.

Notes

HEWLETT
PACKARD

Part No. 09835-10051
Rev. B

Printed in U.S.A.
November 2, 1980