

---

#### HP 9895 Installation Note

Your HP 98228A Disc ROM can control both HP 9885 and HP 9895 disc drives. The ROM assumes a default device (bus) address of 07 for the 9895 Disc Memory. Since each 9895 is set at bus address 01 when shipped, please change the bus address to 07 during installation. The bus address switch is located behind the 9895 front panel, as shown in the 9895 User's Manual.

---



**HP Computer Museum**  
**[www.hpmuseum.net](http://www.hpmuseum.net)**

**For research and education purposes only.**



# HP 9825 Desktop Computer Disc Programming

Manual Part No. 09825-90220

Microfiche No. 09825-99220



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

Copyright by Hewlett-Packard Company 1980

## Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition 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.

August 1980...First Edition (replaces 09885-90000)

---

### NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

# Table of Contents

## Chapter 1: Disc Drives & Formats

Introduction .....	1-1
ROM Installation and Memory Usage .....	1-1
HP9885 Disc Drives .....	1-1
HP9895 Disc Memories .....	1-2
Flexible Disc Media .....	1-3
Flexible Disc Structure .....	1-4
Data Compatibility .....	1-5
File Directory .....	1-5
Disc Interleave .....	1-6
Disc Backup and Error Recovery .....	1-8
Disc ROM Bootstraps .....	1-8
Disc File Structure .....	1-9
Serial File Access .....	1-10
Random File Access .....	1-11
Syntax Conventions .....	1-12
Addressing Disc Drives .....	1-13
Drive Statement (drive) .....	1-14
Disc Type Function (dtype) .....	1-15
Cataloging Disc Files .....	1-16
Catalog Statement (cat) .....	1-16
Rename Statement (renm) .....	1-17
Kill Statements (kill, killall) .....	1-18
Disc Error Messages .....	1-19
HP-IB Programming Considerations .....	1-20

## Chapter 2: Non-data Disc Files

Introduction .....	2-1
Save Statement (save) .....	2-2
Get Statement (get) .....	2-4
Chain Statement (chain) .....	2-7
Save Keys Statement (savek) .....	2-9
Get Keys Statement (getk) .....	2-9
Resave Statement (resave) .....	2-9
Save Memory Statement (savem) .....	2-10
Get Memory Statement (getm) .....	2-10
Get Binary Statement (getb) .....	2-11
Secure Programs .....	2-11

**Chapter 3: Disc Data Files**

Introduction .....	3-1
Using Disc Data Files .....	3-2
Open Statement (open) .....	3-2
Files Statement (files) .....	3-3
Data File Pointers .....	3-4
Assign Statement (asgn) .....	3-5
Serial Print Statement (sprt) .....	3-7
Serial Read Statement (sread) .....	3-10
Random Print Statement (rprt) .....	3-12
Random Read Statement (rread) .....	3-15
Positioning the Pointer .....	3-16
On End Statement (on end) .....	3-19
Type Function (type) .....	3-20
Data Storage Requirements .....	3-22
Summary of EOR and EOF Marks .....	3-23

**Chapter 4: Disc Utilities**

Introduction .....	4-1
Initializing Discs (init) .....	4-2
Initializing 9885 Discs (98217A ROM) .....	4-2
Initializing Discs (98228A ROM) .....	4-3
Killall Statements (killall) .....	4-4
Boot Statement (boot) .....	4-4
Verify Boots Statement (vfyb) .....	4-4
Repack Statement (repk) .....	4-5
Verify Statements (von, voff) .....	4-6
Copy Statements (copy) .....	4-7
Disc Copy .....	4-7
File Copy .....	4-7
Partial File Copy .....	4-10
Additional Uses for Copy .....	4-11
Dump Statement (dump) .....	4-11
Disc Dump .....	4-12
File Dump .....	4-12
Load Statement (load) .....	4-13
Disc Load .....	4-13
File Load .....	4-13



98217A Error Recovery Routines

(dirc, dtrk, ltrk, tinit) ..... 4-14

Error d5 Recovery ..... 4-15

Error d6 Recovery ..... 4-16

Error d7 Recovery ..... 4-17

Binary Error Codes ..... 4-18

**Appendix: References**

Disc ROM Syntax ..... A-1

Disc Error Codes ..... A-7

Subject Index ..... A-9





# Chapter 1

## Disc Drives and Formats

### Introduction

The HP9825 Desktop Computers have add-on language ROMs (read-only memories) for controlling HP9885 and HP9895 disc drives. The HP98217A Disc ROM allows any 9825 to control 9885 drives. The HP98228A Disc ROM provides control of both 9885 and 9895 drives with the 9825T Computer.

This manual describes the disc control operations available with each disc ROM. The manual replaces the 9825A Calculator Disc Programming manual, part no. 09885-90000.

### ROM Installation and Memory Usage

Follow the directions in your 9825 Operating and Programming manual when installing a disc ROM. The 98217A ROM can be plugged into any 9825 Computer. The 98228A ROM, however, can be used only with a 9825T.

The 98217A ROM uses 1140 bytes of computer read/write memory. The 98228A ROM uses 1172 bytes of memory.

### HP9885 Disc Drives

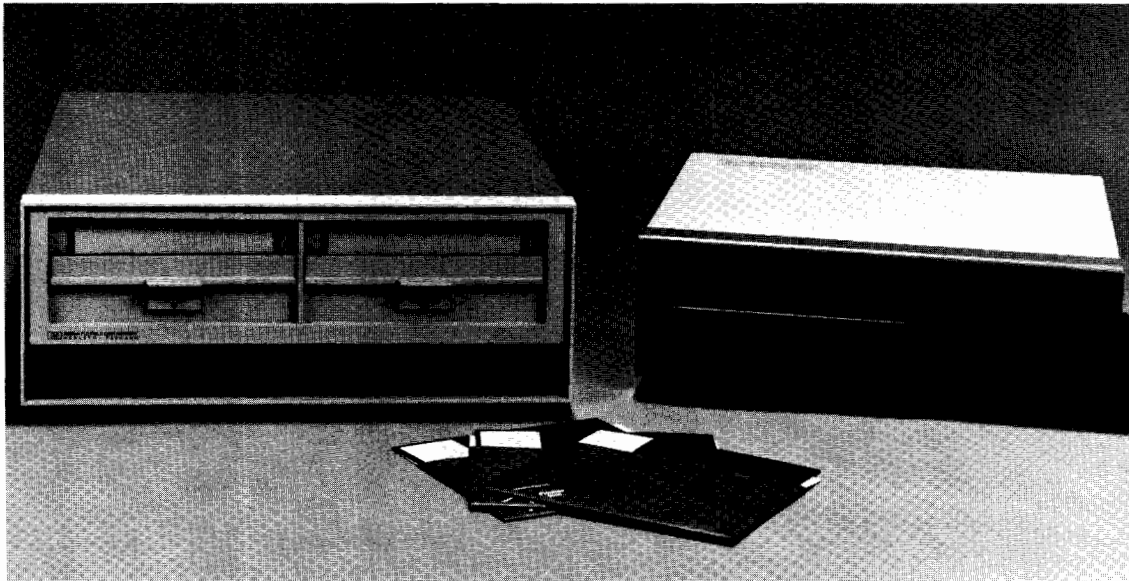
The 9885M (master) drive handles one single-sided, double-density flexible disc and can control up to three 9885S (slave) drives. Each 9885S handles one single-sided disc. The 9825 Computer can control up to eight 9885M drives. (An HP9878A I/O Expander is needed to connect more than three devices to the computer.) 9885 Drives can be controlled with either disc ROM.

Refer to the 9885 Installation Manual, part no. 09885-90010, for details on installing and testing each 9885 drive. The manual also explains how to care for your flexible discs.

## HP9895 Disc Memories

The 9895 Disc Memory is a random-access device which handles double-sided, double-density flexible discs. The standard 9895 has two drives; the 9895 Option 010 has one drive. Each 9895 connected directly to the computer can control one additional 9895 memory, for a total of four drives via one 9895 controller. The 9895 can also handle single-sided discs. The 98228A Disc ROM in a 9825T Computer is required to control 9895 drives.

Refer to the 9895 Flexible Disc User's Manual for instructions on installing and using the disc drives.



HP9895 and 9885 Disc Drives



## Flexible Disc Media

The storage medium for the 9885 and 9895 is a flexible plastic disc 200mm (about 8 inches) in diameter. The disc is enclosed in a protective jacket which has openings for the drive spindle and read/write head access. A notch in the jacket is used to indicate when the disc is write-protected (recording on the disc is not allowed).

Two types of flexible discs (or diskettes) are in current use. The 9885 drives can use only single-sided discs (data is recorded only on the bottom side). Data is recorded on the disc in a "double-density" technique, enabling about 1/2 megabyte of data per disc.

The 9895 drives handle both single-sided discs and discs labelled "double-sided, double-density". The 9895 drive records on both sides of double-sided discs, storing up to 1.2 megabytes of data. Double-sided discs cannot be used in 9885 drives.

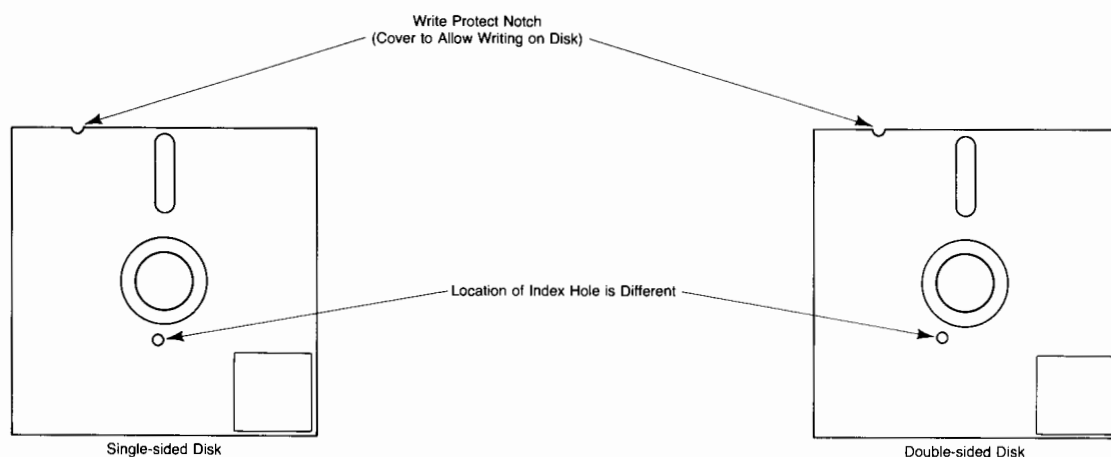
Each flexible disc must be initialized before use. The initialization procedure marks recording tracks on the disc, checks for defects (bad tracks), and establishes a file directory on the disc. Refer to Initializing Discs in chapter 4 for more details.

---

### CAUTION

Only flexible discs approved by HP should be used in the 9885 and 9895 drives. Repeated use of other discs could damage the drive's read/write heads. (Repeated use of single-sided discs in a 9895 drive could also damage the drive.) Contact an HP sales office for a list of approved discs.

---

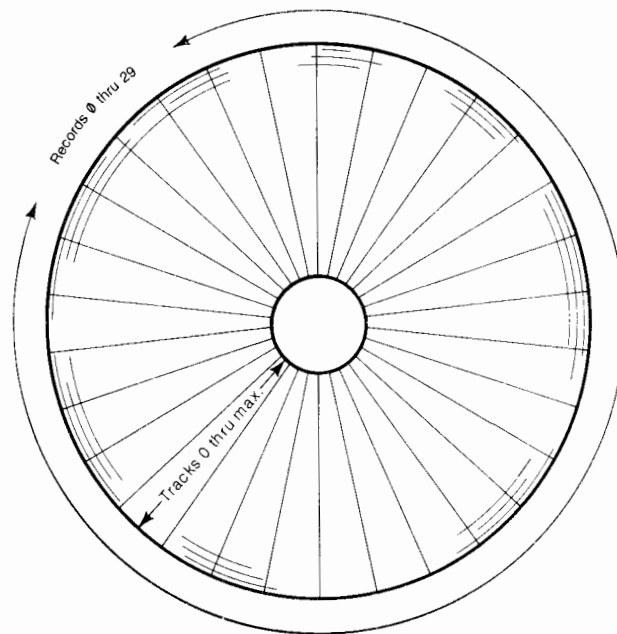


Identifying Flexible Discs

## Flexible Disc Structure

Data is stored magnetically on the disc as binary digits or bits. The bits are stored (written) and retrieved (read) via a read/write head which contacts the disc surface. The 9885 drive has one read/write head, while the 9895 drive has two heads, one contacting each side of the disc.

Data is stored in concentric tracks on the disc. The 9885 writes 67 tracks on each disc. The 9895 writes 77 tracks on each side of a disc. Each track is subdivided into sectors or records. The initialization routine establishes 30 records in each track. Each record can hold 256 bytes of data. This is the smallest amount of data which the 9825 Computer can read or write on disc.



Flexible Disc Structure

The number of tracks available on each disc depends upon the drive, the disc ROM used and the condition of the disc. For example, the 98217A ROM initialization routine marks 67 tracks, but uses tracks 0 thru 5 for housekeeping (file directory, bootstraps, etc.). Up to six tracks can also be defective before the routine rejects the disc. So the useable disc area is from 55 to 61 tracks.

The number of useable tracks available with each ROM/drive combination is summarized next.

**Tracks Available for User Storage**

Initialization Routine	9885 Drive	9895 Drive	
		(single sided)	(double sided)
98217A ROM	55-61	---	---
98228A ROM	59-65	71	148

The 98228A initialization routine ensures that 71 or 148 tracks are available by marking 77 tracks on each side of a disc and reserving four tracks as spares. The main and spare directories are placed on tracks 0 and 1.

## Data Compatibility

Since the same single-sided media can be used in both the 9885 and 9895 drives, single-sided discs recorded via the 9825 Computer on one drive can be read via the other. Discs recorded via other computers, however, may not be usable with the 9825. The data file format is compatible with HP9835 and 9845 Desktop Computers. Data files from HP250, HP300 and HP1000 flexible discs are also compatible. Although the 9895 can read single-sided discs recorded in IBM 3740 format, those discs cannot be read or written into with the 9825 Computer. The dtype function allows identifying discs initialized in IBM format. See the end of this chapter for details on the dtype function. HP cannot guarantee data compatibility with any other discs or formats.

## File Directory

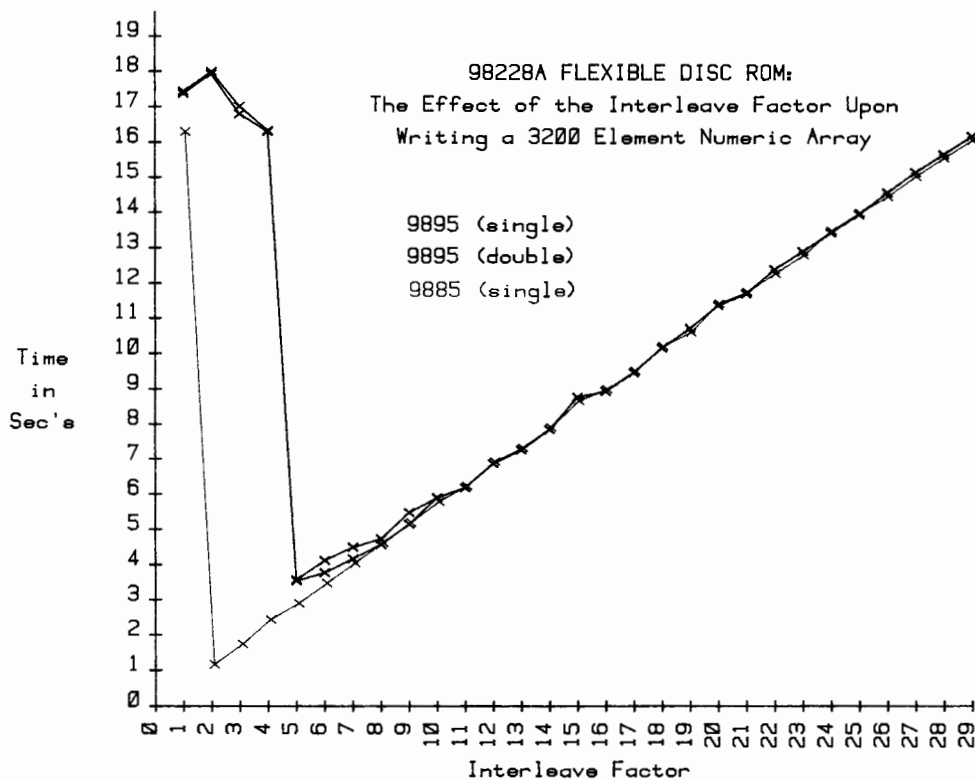
Track 0 on each disc is reserved for a directory of file names and locations on the disc. Track 0 also contains a table of available free spaces on the disc. This table is automatically updated each time a file is created (opened) or purged (killed) from the disc. The 98217A initialization routine creates a backup on track 5; the 98288A routine places the backup on track 1. See the 98217A Disc ROM Bootstraps section for more details.

## Disc Interleave

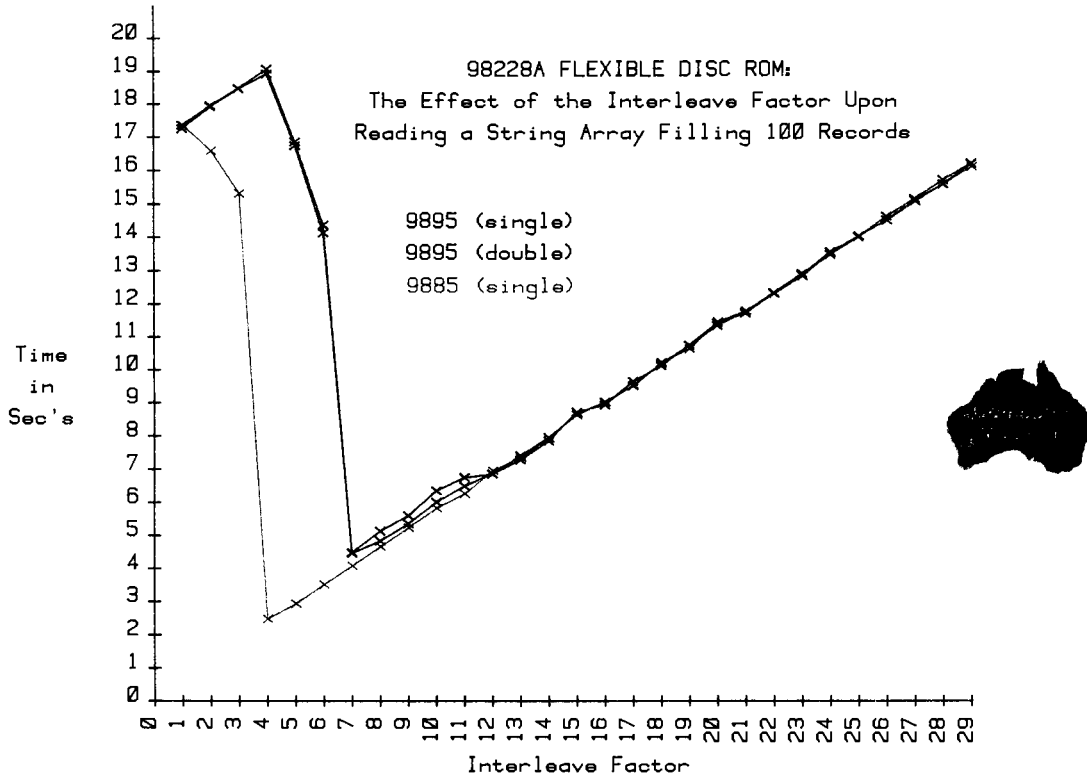
For some applications, it may be advantageous to initialize a disc using an alternate record interleave format. The interleave format determines the number of disc revolutions needed to read a complete track. Adjusting the interleave allows fine-tuning the disc read/write time to the time needed for the computer to handle the data.

The 9885 interleave is fixed with the 98217A ROM to a factor of 2. With the 98228A ROM, however, the interleave factor can be specified during disc initialization. The range is an integer from 1 thru 29. The default interleave for 9895 discs is 5; the default for 9885 discs is 2.

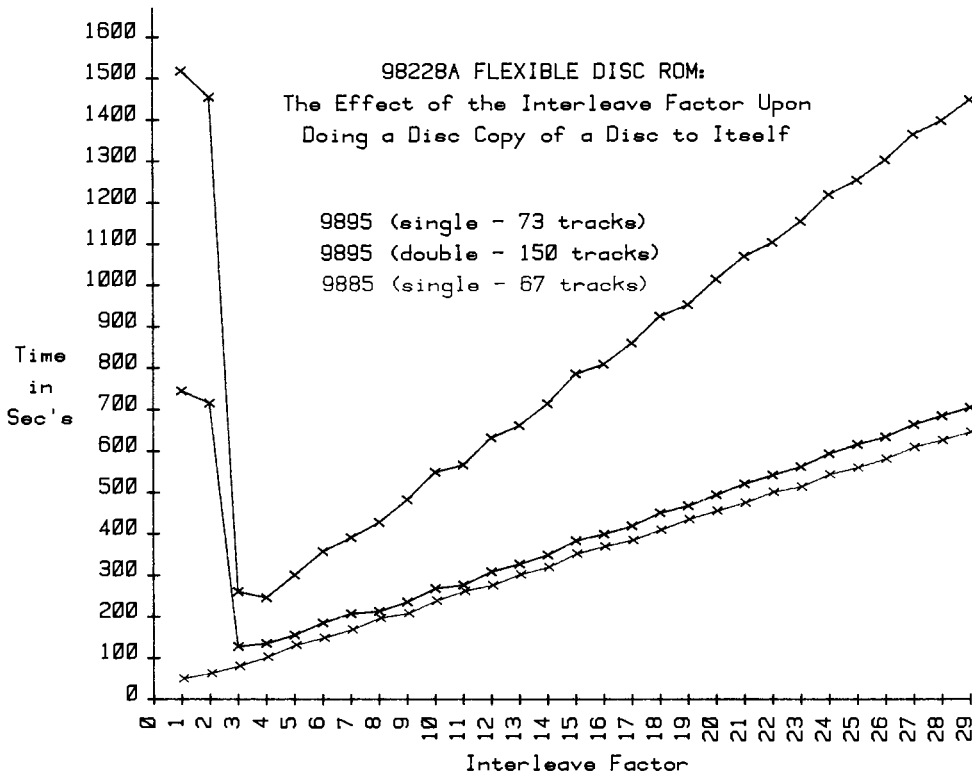
The following graphs show read/write times for typical disc operations with a 98228A ROM. The first graph compares the time needed to write a 3200-element numeric array to each disc. In general, the default interleave settings (2 for 9885 and 5 for 9895) offer optimum performance for this and other disc operations.



The next graph shows an exception to using the default interleaves, reading long strings from disc. Here, an interleave of 4 works best with a 9885, while 7 provides the best performance with a 9895.



The last graph shows the effect of interleave on copying discs. In each case, an entire disc was copied to an identical medium. Note that an interleave of 1 is best for a 9885, but 3 is best for a 9895 drive. When backing up discs, remember that the source disc interleave need not match the destination disc interleave. Better performance can be had when doing disc copies by just initializing the destination disc to the optimal interleave.





## Disc Backup and Error Recovery

Since flexible discs can hold a tremendous amount of information, each disc's contents should periodically be copied or backed up to another disc. This insures against loss of data due to disc wear or a drive malfunction.

Disc drive operation is similar to that of a precision record player. Each time a record is played, the needle causes some record wear. After hundreds of playings, measurable distortion occurs. Similarly, the disc drive's read/write head(s) cause some wear each time a disc file is accessed. When the disc is too worn to be read, error messages indicate accessing the backup file directory or a read failure. When either of these errors occurs, it's past time to backup the disc.

The disc copy statement is provided to backup the contents of one disc to another of the same type. If a second drive is not available, use the dump and load statements to temporarily store data on a tape cartridge (dump) and then record it on another disc (load). See chapter 4 for more details.

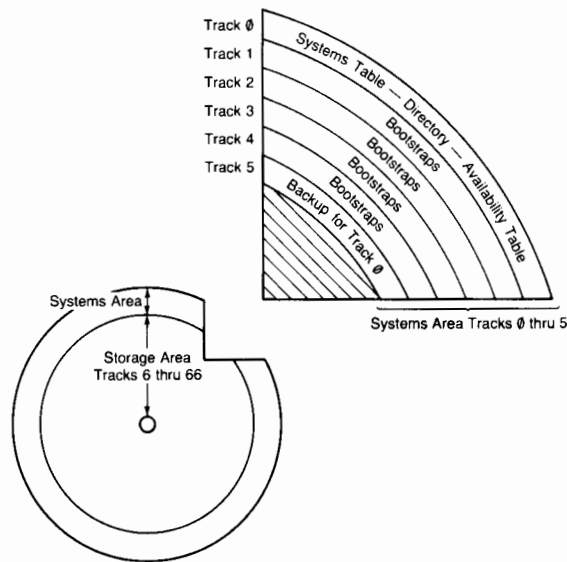
## 98217A Disc ROM Bootstraps

Most of the disc control routines for the 98217A ROM are stored on tracks 1 thru 4 of each disc it has initialized. These routines are loaded or "bootstrapped" into read/write memory when a disc statement is stored or executed. Access to these bootstraps requires that an initialized disc (with bootstraps) be loaded and the drive door closed. If not, an error is issued when the operation is attempted; the 98217A ROM cannot access a disc without bootstraps.

Since the new 98228A ROM has eight times the memory of the 98217A ROM, the new ROM has its bootstraps built in. This means a disc need not be installed when you store disc statements. Also, the new ROM can access discs with or without bootstraps; the new ROM ignores them.

The 98217A bootstraps are written on disc during initialization from a 9825/9885 disc system cartridge. 9885 discs initialized from a system cartridge prior to revision E, however, are not compatible with the 9825T Computer using a 98217A ROM. These older discs must be "re-booted" from a system cartridge having revision E or later boots. Booting a disc does not affect stored data. Revision E boots are compatible with any 9825/98217 ROM system. The boot statement is covered in the Utilities chapter.

To determine what level boots are on a disc, insert the disc and do a cat with the 98217A ROM. The revision level is listed in the catalog header. To determine the revision level of a disc system cartridge, either initialize or boot a disc using the cartridge, and then do a cat of that disc.



98217A Bootstraps Area on a 9885 Disc

## Disc File Structure

The flexible disc system is based on user-defined storage areas called files. A disc can hold up to 352 files, depending on file sizes. Files can be defined to hold numeric and string data (type D files), programs (P files), computer memory (M files) and special function key definitions (K files). You can also load binary programs from a prerecorded type B file. The catalog statement allows identifying files on each disc. You create these files and, for data files, specify their size. The statements and functions covered in the next chapters also allow you to store information in, and retrieve information from, your disc files.

Each file contains one or more whole records; a record is never split between files. Each record is 256 bytes long, and is the smallest addressable unit of data on the disc. The size of a program, key or memory file is automatically set when the information is stored on disc. You specify the size of your data files.

There are two ways to access data files on disc: serially and randomly. Although data files can be printed and read using either method, one method is usually selected depending on the application. The following paragraphs explain the advantages and disadvantages of using each method to help you select the right one for your application.

## Serial File Access

When you wish to handle data in a sequential order, rather than in a selected (random) order, store it using serial print (sprt) statements and retrieve it using serial read (sread) statements. An sprt stores data compactly in a file, beginning at the start of the file and continuing through as many records as needed to print all items in the sprt data list. An end-of-record (EOR) mark is written after the last item. If another sprt statement follows the first, the data is printed starting at that point in the file by printing over the first EOR. So data is printed compactly, one item after another, with no identifiable marks in between.

As data is serially written, any previously-written EORs, or end-of-file (EOF) marks are simply written over. However, if an EOF is encountered where an EOR it to be written at the beginning of a record, the EOF is left in place of the EOR. A subsequent serial print would write over the EOF, as if it were an EOR.

Data serially printed in a file can be retrieved by using a serial read (sread) statement. Reading starts at the beginning of the file and continues until all variables in the sread data list are filled. Any EOR marks encountered cause the read to skip to the next record.

Another sread or sprt operation using the file would begin at the point where the last one left off. The beginning of a file is the only point where serial access is possible. So if you wanted to only read the fifth item in a serial file, you'd have to read the first five items into variables using sread.

## Random File Access

When you want to store and retrieve blocks of data in a random sequence, you can access individual records within each file using random print (rpri) and random read (rread) statements. Each record is accessed by a unique record number. The number of records in a file was specified when the file was created (opened). An rpri stores data beginning at the start of a specified record. Up to 256 bytes of data, including some overhead explained in chapter 3, can be printed in each record. Data is not allowed to overflow to the next record.

A rread statement starts reading at the beginning of a specified record. The length of the rread variable list determines how much of the record is read. Another rread to the same record would start again at the beginning of that record. An error is issued if an EOR or EOF is encountered before the data list is filled.

Random file access may not use disc space as efficiently as serial access, since records are may not be filled as completely. This waste can be minimized or even eliminated by partitioning the data so each record is filled as completely as possible.

Access to individual items is generally faster using random access, since the drive can seek almost any record on disc at the same speed.

**Comparison of Data Access Methods**

	Serial Access	Random Access
<b>Access Time:</b>	Varies - longer for data at end of file.	Good - direct access to any record.
<b>Storage Efficiency:</b>	Best - data items are printed sequentially with no wasted space.	Varies - remainder of each record is not useable.
<b>Ease of Use:</b>	Best when data is always accessed in exactly the same order.	Best when handling data randomly, with individual records.

## Syntax Conventions

The following conventions and terms are used in describing each disc ROM statement and function.

**dot matrix** - all keywords and characters in `dot matrix` must appear as shown.

[ ] - parameters appearing in brackets are optional.

**constant** - a fixed numeric value, like 2.45.

**text** - characters within quotes, like "string".

**expression** - a numeric constant (like 2), a variable (A or r3) or a numeric expression (like B+2.5).

**string expression** - text within quotes (like "hello"), a string variable name (like A\$) or a string expression like upc (A\$).

**drive number** - a numeric expression from 0 thru 3 specifying the disc drive. The drive is set to respond to the number via a switch setting.

**select code** - a numeric expression specifying the address set on the disc drive's interface card. The 98217A ROM allows an integer from 8 thru 15 for the 9885M; select code 8 is default.

The 98228A ROM allows a one- thru four-digit integer select code. A one- or two-digit integer (8 thru 15) specifies a 9885M drive's interface select code. A three- or four-digit number of the form ssdd specifies a 9895 drive, where ss is the HP-IB interface's select code (from 2 thru 15) and dd is the drive's HP-IB device address (from 00 thru 07). Select code 0707 is default. See the Drive Statement (next) for examples.

**file name** - a string expression or text specifying the name of a disc file. The name can be up to six characters, but cannot contain quotes, commas, semicolons, colons, blanks, or ASCII characters less than octal 41. Each file on a disc must have a unique name. The String ROM is needed to use string expressions.

**file number** - an integer expression from 1 thru 10 specifying an assigned data file.

**line number** - a numeric expression from 0 thru 9999 specifying a program line.

**buffer name** - a string expression or text specifying the name of a data buffer (Extended I/O ROM).

**record number** - a numeric expression specifying an individual record of information (data files only).

## Addressing Disc Drives

All disc operations except copy, init and killall are automatically addressed to a default disc drive. The address consists of the disc controller interface's select code, the disc drive number and, with the 9895, the controller's HP-IB device address (0 thru 7). The default drive's address is specified via the drive statement explained next.

The drive address parameters correspond to switch settings on the drive and its interface card. The 9885M drive is set at the factory to respond to drive number 0, while the 9885S is set to drive number 1. The 9885M interface (98032A option 085) is set to select code 8. So the 98217A ROM default address 0,8 automatically addresses a 9885M as shipped from the factory.

The 9895 drive is supplied from the factory set to respond to device address 00. The 98034A HP-IB interface is supplied set to select code 7. Since the 98228A Disc ROM default address is 707, however, be sure to change the 9895 device address to 07 before executing other disc operations. The 9895 User's Manual explains setting the device address.

## Drive Statement

```
drive drive number [ ; select code]
```

The drive statement specifies the disc drive to be accessed by succeeding disc operations. The drive number can be from 0 thru 3; drive number 0 is default with each disc ROM. The select code is covered on the preceding page. Select code 8 is default with the 98217A ROM, and 707 is default with the 98228A ROM.

```
drive 1;9      Specifies 9885 drive 1 on interface select code 9.
```

or

```
1+D;9+S  
drive D;S
```

```
drive 2;701    Specifies 9895 drive 2 via the 9895 controller at HP-IB device address  
01. The HP-IB interface is set to select code 7.
```

```
drive 2        Specifies drive 2 on either the previously-specified select code or the  
default select code.
```

Notice that a one- or two-digit select code implies a 9885 drive; a three- or four-digit select code indicates a 9895 drive (98228A ROM only).

The 98217A ROM automatically sets the default 9885 drive address (0,8) at powered up, reset, or after erase a. The 98228A ROM sets the default drive address 0,707.

## Disc Type Function

The 98228A Disc ROM provides a function for returning information on the type of medium in a 9895 or 9885 drive (dtype isn't available with the 98217A ROM).

dtype

The function is addressed to the default drive and returns an integer value:



Value	Meaning
0	Unable to access default disc controller.
1	Drive door is open or drive not present.
2	Drive door closed, but door was opened since last disc operation. File pointers are cleared.
3	9895 drive, single-sided disc, HP format.
4	9895 drive, double-sided disc, HP format.
5	9895 drive, single-sided disc, unknown format.
6	9895 drive, double-sided disc, unknown format.
7	9895 drive, single-sided disc, IBM 3740 format.
8	9885 drive, single-sided disc.

The unknown format could be an uninitialized disc or a disc formatted by another computer. The 9825 cannot read these discs.

A disc identified with IBM format CANNOT be read by the 9825 Computer. The dtype function allows you to identify these discs to avoid errors. If information on an IBM formatted disc is not needed, it can be reinitialized with the HP format; see Initializing Discs in chapter 4.



## Cataloging Disc Files

### Catalog Statement

```
cat [ select code or "buffer name " ]
```

The cat statement prints a list of user files on the default disc. Each catalog entry includes the file name, its type (code) and its current size. These file type codes are used:

- B** Binary-program file.
- D** Data file (record length = 256 bytes).
- K** Special function keys file.
- M** Memory file.
- O** Other file (not created by 9825).
- P** HPL program file.

The catalog expresses the size of most files in bytes (B). Data files, and type other files with the 98228A ROM, are expressed in records (R).

Specifying cat0 or omitting the select code outputs the catalog to the internal printer. Specifying cat16 outputs a more-detailed catalog to the internal printer. Here are examples of each catalog using the 98228A ROM:

<pre>DRIVE 0,8      SS AVL RCRDS 1585  NAME TYPE SIZE ===== grades D   100  R Memory M 62774B Count  P  112  B Name   P  242  B give   D   10  R files1 P  112  B keys   K  100  B sread  P   84  B dcopy  P  266  B</pre>	<pre>DRIVE 0,8      SS AVL RCRDS 1585  NAME TYPE SIZE #REC TRCK RCRD ===== grades D  100  T2   R0 Memory M 62774B  246  T5   R11 Count  P  112B  1    T13  R17 Name   P  242B  1    T13  R18 give   D  10   T13  R19 files1 P  112B ... ..</pre>
--	--

The catalog header indicates the drive number, select code and disc type: SS for single-sided or DS for double-sided. The 98217A ROM outputs a catalog header indicating the 9885 drive number and bootstrap revision. For example:

```
CAT DRIVE 0 /A
```

Specifying a select code from 2 thru 15, or specifying a buffer name, outputs the more-detailed catalog to the selected device. Here's a catalog output to an HP9866B Printer:

```
DRIVE 0,8      DS
AVL RCRDS 4072
NAME TYPE SIZE      #REC TRCK RCRD
=====
grades D                100   T2   R0
Class P   96B           1    T14  R6
Memory M 62774B       246   T5   R11
Count P   62B           1    T13  R17
Name P   242B           1    T13  R18
give D    10            10   T13  R19
files1 P  112B          1    T13  R29
keys K   100B           1    T14  R0
sread P   84B           1    T14  R1
dcopy P  266B           2    T14  R4
```

For details on using I/O buffers, refer to your 9825 I/O Control Reference or Extended I/O Programming manual. An example application using cat and an I/O buffer to read cataloged information is in the Copy Statements section of chapter 4.

## Rename Statement

```
renm old file name , new file name
```

The renm statement changes the name of an existing disc file. For example, to change the name of a file named Master to MASTER, execute:

```
renm "Master" , "MASTER"
```

## Kill Statements

Two statements are available for purge user disc files. Each operation purges only the specified file(s), without affecting disc initialization. The kill statement purges a specified file on the default disc:

```
kill file name
```

Some examples are:

```
kill "Count"
```

```
1: ent "Enter file to be purged",N$
2: kill N$
```

The killall statement purges all user files. The 98217A ROM uses a binary program which must be loaded from the Disc System Cartridge (execute ldb0). The syntax is:

```
killall
```

The operation is addressed to the default disc drive.

The killall statement is built into the 98228A ROM. The syntax is:

```
killall drive number , select code
```

The parameters are identical to those in the drive statement, but the select code is not optional with killall. Here are some examples:

<pre>killall 0,8</pre>	Purges all files on 9885 drive 0 via select code 8.
<pre>killall 1,706</pre>	Purges all files on 9895 drive 1, via the 9895 controller at device address 06 via the HP-IB interface set to select code 7.

A one- or two-digit select code implies a 9885 drive, while a three- or four-digit select code addresses a 9895 drive.

## Disc Error Messages

Each disc ROM issues error messages indicating when the disc controller cannot be accessed or when an improper operation is attempted or when a hardware error occurs. These errors are listed at the back of this manual. Some errors can be trapped via the on err (on error) statement; see the 9825 Extended I/O Programming Manual or 9825 I/O Control Reference. Errors d0 thru d9 are disc drive hardware errors; d5 thru d7 are “recoverable”, as explained in chapter 4. Error F0, file overflow, occurs when reaching the end of file during disc reads and prints. This error can be trapped via the on end statement, as shown in chapter 3.

In addition to numbered error messages, a warning message is flashed on the display when the computer cannot access the disc controller:

DISC IS DOWN (98217A Disc ROM)

UNABLE TO ACCESS DISC CONTROLLER (98228A Disc ROM)

This message is flashed on the display until either:

- The computer can access the controller (e.g., power is restored).
- The STOP key is pressed (error F8 is issued).
- The RESET key is pressed.



Since the 98217A ROM bootstraps are stored on disc, mainframe error 03 (mnemonic unknown) occurs when attempting to store a disc statement when the disc controller is inaccessible or the drive door is open. If the 98217A ROM attempts to access bootstraps from a disc initialized via the 98228A ROM (no boots on disc), the ROM may not issue the DISC IS DOWN message, but hang up the system. If this occurs, press RESET to regain control.

## HP-IB Programming Considerations

HP9895 drives are controlled via an HP-IB interface. When other devices are also connected to the same interface, the programmer should review the following considerations before developing system software. Complete information on HP-IB operation is in the 9825 I/O Control Reference.

1. The 9895 must be interfaced via a "revised" 98034A HP-IB Interface card; an original 98034A Interface cannot be used.
2. A listen-always device, such as a printer in listen-always mode, should not be on the bus with the 9895.
3. Use of an HP bus analyzer should not affect most disc operations. Since both the 9895 and 9825/9895 drivers have timeout loops, however, error d8 (hardware error) could occur if the analyzer slows down certain operations.
4. Any disc statement accessing the 9895 controller clears the bus talker/listener addresses and any interrupt conditions previously set up for that bus. So 9895 control operations should not be intermixed with interrupt-driven operations on the same bus. Applications requiring intermixed 9895 control and interrupt-driven operations should use a separate HP-IB interface for the 9895.
5. A 9895 disc statement will wait if a transfer (tfr) is in progress on the bus. If an end-of-line service interrupt is enabled for a transfer, however, the interrupt may never occur since the 9825/9895 disc drivers may clear all bus interrupt conditions before the specified device has a chance to interrupt. To avoid interference between EOL service interrupts and disc operations, a line preceding each disc statement could check status of the transfer buffer and wait for it to go not-busy.
6. The 9825/9895 system uses parallel polls during disc operations. Since the 9895 may or may not respond to a parallel poll after a disc operation, the programmer should not expect a meaningful parallel poll bit from the 9895. The 9895 parallel poll bit assigned depends on the disc controller's device address:

HP-IB Address	Parallel Poll Bit
0	7
1	6
2	5
3	4
4	3
5	2
6	1
7	0

7. The 9895 controller reads its device address thumbwheel switch to determine its bus address only after certain operations. If the device address is to be changed, be sure do one of these operations immediately after setting the switch:
  - Switch the 9895 controller off and on again.
  - Issue a universal device clear (clr).
  - Issue a selective device clear to the old address.
8. If the 9825/9895 disc driver cannot access the 9895 controller, the driver attempts to clear the disc controller by issuing a selective device clear to the disc controller's assumed address.
9. When a serial poll sequence on the bus has been aborted without a serial poll disable command, the 9825/9895 disc driver will sense the condition and issue the SPD command immediately before the next disc operation.



# Chapter 2

## Non-data Disc Files



### Introduction

Disc files can be created to hold programs, data, special function key definitions and the computer memory. This chapter shows how to create and access all but data files. The operations are the same with each disc ROM and each disc drive. Each operation accesses the default disc drive, assigned via the drive statement (see chapter 1).

All disc operations covered here can be executed from either the keyboard or a program. Except where noted, each can also be executed in the live keyboard mode.

These operations are available for non-data files:

<code>drive</code>	Assigns the default disc drive (see chapter 1).
<code>cat</code>	Lists the files on the disc (see chapter 1).
<code>save</code>	Stores program lines from memory into a disc file.
<code>get</code>	Loads program lines from a disc into memory.
<code>chain</code>	Same as <code>get</code> except program variables are not erased.
<code>kill</code>	Purges a specified file from the disc (see chapter 1).
<code>renn</code>	Renames an existing file (see chapter 1).
<code>resave</code>	Stores program lines into an existing program file.
<code>savek</code> <code>getk</code>	Stores and loads special function key definitions.
<code>savem</code> <code>getm</code>	
<code>getb</code>	Loads binary programs from a prerecorded binary file.



## Save Statement

The `save` statement stores an entire program or part of it in a specified file.

```
save file name [ : 1st line number [ : 2nd line number]]
```

With a program in the calculator's memory, execution of the `save` statement stores the program on the disc. The specified program file takes up the number of whole records needed for the program.

The optional line number parameters enable you to store part of your program rather than all of it. With one line number specified, the `save` statement stores only the lines after (and including) the specified line. With both parameters, the lines between (including the specified lines) are stored on the disc. Your whole program is still present in the calculator, whether all or only a portion of it is saved on the disc.

To illustrate use of the `save` statement, key in the following program.

```
0: ent J
1: for I=J to J+
  10
2: prt I
3: next I
4: end
```

You can store the previous program in a file named `Count`, for example, by executing this statement from the keyboard:

```
save "Count "
```

A squaring program is added to the original program just saved:

```

0: dsp "Counting
   Program";wait
   1000
1: dsp "For squa
   ring, key in
   100";wait 1000
2: dsp "then
   the number to
   be squared.";
   wait 1000
3: ent "Key in
   an integer";J
4: if J>99;to 9
5: for I=J to J+
   10

```

```

6: prt I
7: next I
8: end
9: dsp "Squaring
   Program";wait
   1000
10: ent "Key in
   another integer
   ";K
11: for L=K to
   K+10
12: prt L,L^2
13: next L
14: end

```

Lines 0 thru 8 print ten consecutive numbers from the first number entered. Lines 9 thru 14 print ten consecutive numbers and their squares, from the second number entered.

Once this program has been keyed into the calculator, you can save it in a file named Master, for example, by executing this statement from the keyboard:

```
save "Master"
```

You can also store the second half of the program in a file named Square, for example, by executing this statement from the keyboard:

```
save "Square",9
```

Finally, to store the first half of this program in a file named First, for example, execute the following statement from the keyboard:

```
save "First",0,8
```

You now have four programs stored using different file names. Once a program is stored on the disc, it can be loaded back into the calculator memory using the `set` or `chain` statements.

## Get Statement

The `get` statement loads program lines from a type P file into the computer. All variables are erased.

```
get file name [ : 1st line number [ : 2nd line number ]]
```

Whenever the `get` statement is executed, all program lines in the specified file are loaded into memory. All program lines previously in the calculator memory are erased except those lines preceding the first line number, if one is specified. If the second line number is included, program execution automatically begins at that line number.

For example, execute this statement to load the previously-saved program named Master back into the computer:

```
get "Master"
```

```
0: dsp "Counting
   Program" : wait
   1000
1: dsp "For squa
   ring, key in
   100" : wait 1000
2: dsp "then
   the number to
   be squared." :
   wait 1000
3: ent "Key in
   an integer" : J
4: if J > 99 : goto 9
5: for I = J to J +
   10
6: prt I
7: next I
8: end
9: dsp "Squaring
   Program" : wait
   1000
10: ent "Key in
   another integer
   " : K
11: for L = K to
   K + 10
12: prt L, L + 2
13: next L
14: end
```

The program can be altered once it is in the calculator, but the information on the disc remains unchanged (unless you change it using a `replace` statement).

From within a running program, if the first line number is specified in a `set` statement, the program is loaded beginning with that line number and is renumbered\* from that line number. Program lines with numbers lower than the first line number are retained in memory; all other lines previously in memory are erased. If the second line number is not included, the default value becomes the first line number and program execution automatically begins at the first line number specified from within a running program only.

The following statement, in addition to loading the program into the calculator, renumbers the lines, starting with line 15.

```
set "Master",15
```



A listing of the calculator memory at this point is shown below.

```
0: dsp "Counting
   Program";wait
   1000
1: dsp "For squa
   ring; key in
   100";wait 1000
2: dsp "then
   the number to
   be squared.";
   wait 1000
3: ent "Key in
   an integer";J
4: if J>99;ato 9
5: for I=J to J+
   10
6: prt I
7: next I
8: end
9: dsp "Squaring
   Program";wait
   1000
10: ent "Key in
   another integer
   ";K
11: for L=K to
   K+10
12: prt L;L+2
13: next L
14: end
```

(continued)

\*Any `ent` statement addresses (line numbers) in the program are not automatically renumbered.

```

15: dsp "Countin
a Program";wait
1000
16: dsp "For
saudring, key
in 100";wait
1000
17: dsp "then
the number to
be saudred.";
wait 1000
18: ent "Key in
an integer",J
19: if J>99;ato
9
20: for I=J to
J+10
21: prt I
22: next I
23: end
24: dsp "Saudrin
a Program";wait
1000
25: ent "Key in
another integer
",K
26: for L=K to
K+10
27: prt L,L+2
28: next L
29: end

```

As you can see, the program originally loaded has not been erased; the second program has been loaded after it (beginning at line 15). Had the second program been renumbered from line 5, only lines 0 thru 4 of the original program would have remained.

When a second line number is included, the `set` statement causes program execution to begin immediately at the specified line number. For example:

```
set "Master",15,9
```

By executing the statement above, the program is renumbered from line 15 again, and automatically executed beginning at line 9.

## Chain Statement

The `chain` statement is identical to the `set` statement discussed previously, except that current variables are not lost. When executed from the keyboard, the loaded program will not be automatically run unless the second line number is specified, as with the `set` statement.

```
chain file name [ : 1st line number [ : 2nd number]]
```

The `chain` statement is most often used in a program to link a program previously stored on the disc to one currently in the memory.

The following three programs are used to illustrate the `chain` statement and how the values of variables are retained.

First key in and save this program:

```
save "Begin"
0: prt "Begin
  executed", ""
1: chain "Middle"
  ",0,0
2: end
```

Then key in and save this program:

```
save "Middle"
0: 0+J
1: sto 3
2: 2+J
3: if J=1;sto 7
4: if J=2;sto 9
5: prt "Middle
  chained to
  Begin and
  executed from
  0", ""
6: chain "End",
  0,0
7: prt "Middle
  chained from
  End and exec
  uted from 3", ""
8: chain "End",
  10,2
9: prt "Middle
  chained from
  End and exec
  uted from 2", ""
10: end
```

Now key in and save the last program:

```
save "End"
```

```
0: if J=2:eto 14
1: prt "End chained fromMiddle
and executed
from 0",""
2: 1→J
3: chain "Middle",0,3
4: prt "End chained fromMiddle
and executed
from 3",""
5: end
```

Run the programs by executing:

```
set "Begin",0,0
```

In these programs, the values of variable J are retained when chaining from program to program, causing selected portions of the last two program segments to be run.

Here's the printout:

```
Begin executed

Middle chained
to Begin and
executed from 0

End chained from
Middle and executed
from 0

Middle chained
from End and
executed from 3

Middle chained
from End and
executed from 2

End chained from
Middle and executed
from 3
```

\*The `eto` statement addresses are not automatically renumbered. Therefore, the `eto` address in line 0 refers to a line number in the program after its lines have been renumbered. To avoid this situation, use labels instead of line numbers as `eto` statement addresses.

## Save Keys Statement

The save keys (save k) statement stores special function key definitions in a specified file.

```
savek file name
```

The definitions of all 24 special function keys can be stored in one file at the same time.

## Get Keys Statement

The get keys (get k) statement loads special function key definitions from a specified disc file.

```
getk file name
```

When `get k` has been executed, the original information is returned to the special function keys. All of the special function keys can then perform the same operation they did when their definitions were saved in the key file.

In addition, when `get k` is executed, previously defined variable values are not lost, similar to the `chain` statement.

## Resave Statement

The `resave` statement enables you to store all or part of a program, which has been modified, back in the same file.

```
resave file name [ # 1st line number [, 2nd line number]]
```

First and second line numbers are used to indicate the portion of the program being resaved as in the `save` statement. This allows you to store a modified program in one step instead of two.



## Save Memory Statement

The save memory (`saveM`) statement stores the calculator's entire read/write memory (program, data, keys, pointers, etc.) in a disc file. A file large enough to store the memory is automatically allocated when the save memory statement is executed.

```
saveM file name
```

To store the calculator's memory in a file named Memory, for example, execute:

```
saveM "Memory"
```

## Get Memory Statement

The get memory (`getM`) statement loads a previously recorded memory file (read/write memory) from the disc and returns the calculator to the state when save memory was executed.

```
getM file name
```

To restore the calculator's memory file just saved using the save memory statement, execute:

```
getM "Memory"
```

---

### NOTE

Memory files saved via one disc ROM (e.g., 98217A) cannot be loaded via the other ROM (e.g., 98228A). Error 51 or 52 indicates an attempt to read into a different memory configuration.

---

## Get Binary Statement

The get binary (getb) statement loads a binary-language routine from a binary (type B) disc file to the computer's memory.

```
getb file name
```

Binaries provide a way to load utility routines and programmable statements into memory which are not available in ROM. Binaries can be loaded, but not saved in another file.



## Secure Programs

An HPL program can be secured from being listed or edited by specifying "SE" when it is recorded (rcf) on tape. A secured program can only be loaded into memory, run and copied to another file. To save a secured program on disc, simply load the secured program back from tape and save it in a disc file. A secured program cannot be unsecured.



# Chapter 3

## Disc Data Files

### Introduction

This chapter covers disc operations unique to data files. If you're familiar with program file operations (chapter 2), you already know `cat` and `kill`. Remember that each operation is addressed to either the default disc drive or the disc defined via the `drive` statement.

These data file operations are available:

- `drive` Specifies the default disc drive (see chapter 1).
- `open` Sets up and names each data file.
- `cat` Lists the files on disc (see chapter 1).
- `kill` Purges the named file from the disc (see chapter 1).
- `files` } Assigns a unique number to each specified data file for use with disc read and  
`osgn` } print operations.
- `sprt` } Prints data, either serially or randomly, into a disc file.  
`rprt` }
- `sread` } Reads data, either serially or randomly, from a disc file.  
`rread` }
- `on end` Automatically exits a disc read or print operation when the end of data or file is reached.
- `type` Returns the type of the next data item in a file.

## Using Disc Data Files

Data files allow you to efficiently store both large and small amounts of numeric and string data in disc. You can handle data either sequentially (serially) or as individual items (randomly), as explained in chapter 1. Follow these steps to create and access each data file:

1. Create each new data file using the `open` statement.
2. Then assign each data file a unique number using either the `asgn` or a `files` statement. The program accesses each file by its number, rather than its name.
3. Now you can print (`spnt` or `rpnt`) data into each file, or ...
4. You can read (`sread` or `rread`) data from each file.
5. When each file is no longer needed, use the `kill` statement to purge it.

In addition to the statements just mentioned, `on end` and the `type` function provide additional flexibility when using data files. Each operation is fully covered in this chapter.

## Open Statement

The `open` statement creates a specified size space for a data file with the indicated number of records, assigns it a name, and places an end of file (EOF) mark at the beginning of each record in the file. Any data in the records used by this file is automatically erased when the file is opened.

```
open file name * number of records
```

An `open` statement indicating file name and size must be executed before data can be printed in that data file. Each data file must be assigned a unique name.

The size of a data file is specified in records. Each record is 256 bytes long. The number of records parameter can be an integer expression from 1 thru the largest available space on disc. Error D8 indicates attempting to open a file larger than the largest available space.

Each opened file is available for your use until you purge it via the `kill` statement.

The first statement in the example programs that follow (illustrating data storage and retrieval) is generally an `open` statement. The `open` statement is included only to remind you that data files must be opened before data can be printed in them. It is best to execute the `open` statement from the keyboard, rather than from a program, since `error D5` results when you run the same program (i.e. try to open the same file) more than once.

## Files Statement

The `files` statement indicates which data files are to be used and optionally the number of the drive accessed (for each file name). The files listed are assigned numbers in the order in which they appear. File numbers are convenient labels used to reference specific files in print and read statements (discussed later).

```
files name or *[# drive number][; name or *[# drive number]] [; ...]
```


The position of the file name in the `files` statement determines the file number. For example, in the following statement, `Name` is assigned file number 1 and `Address` is assigned file number 2.

```
files Name,Address
```

Up to ten file names, each with an optional drive number, can be listed using `files`. Unlike other disc statements, the file name in `files` must be specified as an unquoted string; string variables cannot be used in `files`. Also, the optional drive number must be a numeric constant, not an expression.

A single asterisk (\*) can reserve space for a file in a `files` statement. Then an `open` statement is used later to complete the file assignment by specifying the file name.

If a file specified in the `files` statement has not been previously opened, error D4 is displayed when the `files` statement is executed.

When the `files` statement is stored with an error, a flashing cursor is displayed showing the location of the error when the  key is pressed.

The `files` statement enables you to access more than one drive simultaneously. This is useful when your program is on a disc in one drive and data is on discs in other drives. A single `files` statement can access files from different drives when the optional drive number parameter is used, as long as the select code is the same for all drives being accessed.

However, the drive number from the last `drive` statement is still in effect; only by executing a new `drive` statement can the drive number be changed.

The following program illustrates how data from a disc in drive 0 can be read and printed on a disc in drive 1. (Assume the file named give is open and contains data.)

```

0: drive 1
1: open "take",
  10
2: files give:0,
  take:1
3: for I=1 to 10
4: rread 1,I,A,
  B,C
5: rprt 2,I,A,B,
  C,"end"
6: next I
7: end

```

## Data File Pointers

The files statement associates a specified set of data files with an internal set of file pointers. Maintaining a files table in computer memory provides faster data access than referencing the disc directory before each file read or print.

Each file pointer is set to the first record in the file after an asgn or files is executed. The pointer is automatically advanced through the file during file reads and prints. The pointer can be repositioned by using the random read (rread) statement; see Positioning the Pointer.

File pointers are automatically cleared by a variety of operations. All file pointers are cleared by power on, reset, erasea, a files statement or a drive statement specifying a select code. File pointers for a given drive are cleared by switching the drive off or on, opening the door, or executing a disc copy (destination drive pointers), disc load, repk (repack), init (initialize disc) or killall. All pointers for a specified file are cleared when the file is purged (kill or killall).

## Assign Statement

The assign (`asgn`) statement assigns a number (1 thru 10) to a single file name and, optionally allows a different drive number for the file specified. A return variable can also be used for further file information.

```
asgn file name , file number [ , drive number [ , return variable]]
```

The `asgn` statement enables you to use a string variable for a file name and a variable or a numeric expression for a drive number.

The file number specifies the position of the file name in the `files` statement to be referenced in later print or read statements. The number must be a positive expression whose integer value is between 1 and 10. If it's not an integer, its rounded value is automatically taken.

An optional return variable can be used in an `asgn` statement to determine a file's status. This parameter can be a simple or an array variable. For example:

```
0: files Name,*  
Number  
1: asgn "Addres"  
 ,2,0,K
```



In this example, the asterisk in the `files` statement (in the second file position) is assigned the file name `Addres`. (Assume that the data file `Addres` was opened before the `asgn` statement was executed.) Additional `asgn` statements can be placed later in the same program to reassign a different file name to any file position. The `asgn` statement overrides any `files` statement preceding it. An `asgn` statement sets the data file pointer to the first item of the first record in the specified file.

A return variable can be used to determine the status or type of the file. In the previous example, `K` is the return variable. Its value is determined during execution of the `asgn` statement and can be used anytime in the program. The value of the return variable indicates these conditions:

Value of Variable	Meaning
0	file is available and assigned
1	file doesn't exist
2	program file
3	key file
4	file type not defined
5	memory file
6	binary program file
7	file type not defined
8	file number out of range
9	data file, but records not 256 bytes long. 98228A ROM only.



By checking the value of the return variable, you can avoid errors like D4, file not found .

The following program shows how the `osen` statement can be used.

```

0: dsp "This
Program is used
to open" ;wait
1000
1: dsp "new data
files." ;wait
1000
2: dim A$[6];
fxd 0
3: ent "Enter
file name";A$
4: osen A$,1,0,X
5: if X=1;goto 8
6: dsp A$;"was
opened. New
name"
7: goto 3
8: ent "Number
of Records";J
9: open A$;J
10: prt A$;"is
now opened.
Records=";J
11: goto 3
12: end

```

In this example, line 5 instructs the calculator to branch to line 8 (to open the file) if the file name you enter does not exist.

A string variable cannot be used directly as a file name in a `files` statement, although it can be used in an `osen` statement as shown in the previous program.

As shown in the example above, it is not necessary to use a `files` statement before using `osen`. The `osen` statement can be used to set the data pointer to the first item of a specified file without affecting file pointers for any other files previously specified.

## Serial Print Statement

The serial print (`SPRT`) statement is used to store data on the disc. Data is printed serially in the specified file after the last item previously read or printed.

```
SPRT file number, data items [, "end" or "ens"]
```

The file number refers to the number assigned using the `FILES` or `OPEN` statement.

The data items in the serial print statement can be constants, variables, arrays, strings, substrings or text. The length of the list of data items is limited by the length of the HPL line (80 characters) or by the size of the file.

Either "end" or "ens" can be used as the last parameter in a serial print statement. Using "end" causes an end of file (EOF) mark to be written after the last data item printed. If "end" is not included, an end of record (EOR) mark is printed. This makes it easy to find out how much data is stored in a file using the `TYPE` function or `ON END` statement (explained later).

When "ens" (end suppress) is used as the last parameter, the data list is printed without printing either an end of record or end of file mark after it. Because of this, use "ens" with care.

Serial print and read statements can print and read past record boundaries (256 bytes); as many records as necessary are used to store the data listed.

The data file pointer moves through the file as you store or retrieve data items. Print statements overwrite EOF or EOR marks. As data is read (or written), the pointer moves to the next data item.

Here is an example using the serial print statement to record five student's identification numbers and test grades.

```
0: OPEN "I.D.",1
1: OPEN "Grades"
   ,1
2: FILES I.D.,
   Grades
3: FOR I=1 TO 5
4: ENT "Student'
   s I.D.#",X
5: SPRT 1,X,"end
   "
```

(continued)

```

6:  ent "What is
   the next test
   score?";R
7:  spnt 2;R;"end"
8:  next I
9:  end

```

This program can be used to print identification numbers in the file named `I.D.`, and the corresponding grades in the file named `Grades`.

I.D. Number	Grade
1111	88
2222	67
3333	98
4444	81
5555	99

In the previous program, two separate files are used – one for the students' identification numbers and one for their grades. The information can be combined and stored in one file using the following program.

```

0:  open "Scores"
   ,1
1:  files Scores
2:  for I=1 to 5
3:  ent "Student'
   s I.D.#";X;"Gra
   de";R
4:  spnt 1;X;R;
   "end"
5:  next I
6:  end

```

Line 4 prints the I.D. numbers and test scores of the students alternately in the file named `Scores`. Line 4 also places an EOF mark\* after the five sets of data elements are printed.

\*If "end" is a part of the data being stored, two "end" statements must be included in the program line to place an EOF mark where you want it. If "ens" is a part of the data being stored, two "ens" statements must be included.

You can handle student names rather than numbers by using string variables (String ROM). In the last example program, for instance, replace the numeric variable X with string variable N\$ to handle these names:

Name	Grade
Rob Rood	99
Piper Aune	90
Carol Hafford	88
Andrew Jackson	74
Eric Landry	80

Here's the modified program:

```

0: open "Class",
  1
1: dim N$(15)
2: files Class
3: for I=1 to 5
4: ent "Student'
  s Name",N$,"Gra
  de",R
5: sprt 1,N$,R,
  "end"
6: next I
7: end

```

To illustrate use of the "ens" parameter, the grade of 99 is changed to 100 by executing:

```

rread 1,1;sread1,N$

sprt 1,100,"ens"

```

The random read statement (described later) without a data items list is used to position the pointer to the beginning of the file where the correction is to be made. Then the data is read serially to locate the record and data item to be changed by reading and comparing data until the next item is the one to be changed. (In this case the first item is the one to be corrected.) "ens" parameter prevents placing an EOR or EOF after the corrected item. An EOR or EOF would truncate the rest of the data.

## Serial Read Statement

The serial read (`s read`) statement reads numbers and strings serially from the specified file, starting after the last item read or printed.

```
s read file number ; data variables
```

Before you can work with data which has been stored in a file, you must first read the data into the calculator. Remember that you are not erasing the data stored on the disc by reading it. Instead, data is copied using the variables specified.

The program on a previous page is used to print data on the files, I.D. and Grades. To read the data from these files back into the calculator and print the information, use the following program.

```
0: files I.D.,
  Grades
1: s read 1,S
2: s read 2,T
3: prt "I.D.#",S
4: prt "Grade",
  T,""
5: sto 1
6: end
```

In this program, the `files` statement serves two purposes – it establishes the files with their file numbers in the `s read` statement (lines 2 and 3) and it resets the pointers to the beginning of both files before the `s read` statements are executed. Here's the printout that results when the program is run.

```
I.D.#    1111.00
Grade    88.00

I.D.#    2222.00
Grade    67.00

I.D.#    3333.00
Grade    98.00

I.D.#    4444.00
Grade    81.00

I.D.#    5555.00
Grade    99.00
```

When this program is executed, error F0 is displayed because an attempt is made to read data after an EOF mark is reached.

Data printed in the file named `Class` (see the program on a previous page) can also be read back into the calculator. Use this program to print the data.

```
0: dim F#[15]
1: files Class
2: for I=1 to 5
3: sread I,P#,Y
4: prt "Student"
   ,P#,"Score",Y;
   ""
5: next I
6: end
```



Notice that the `sread` statement must specify the types of data (data elements or string variables) in the order in which they were originally stored in the file. Line 3 reads a string variable and then a score. This program can run only when the order of the data in the file named `Class` is known. Here's the printout:

```
Student
Rob Rood
Score      99.00

Student
Piper Aune
Score      90.00

Student
Carol Hafford
Score      88.00

Student
Andrew Jackson
Score      74.00

Student
Eric Landry
Score      80.00
```

The variables into which you read data items do not necessarily have to be the same variables from which you printed the data items in the file. Although the variable name changes (from N\$ and R when stored, to P\$ and Y, when retrieved), the order in which the two data types are accessed and the types themselves are the same.

An EOF mark should be placed at the end of each serial file; a serial read will halt at the EOF and issue error FO (file overflow). This error can be trapped using `on end`, as explained later. If the serial read's variable list doesn't match the data being read, the serial read could skip over any EORs and cause unexpected results. Placing the EOF after the data ensures the a serial read stops at the end of data.

## Random Print Statement

The random print (`rpri`) statement is used to store individual data items in specified records within a file.

```
rpri file number ; record number [ ; data items[ ; "end" or "ens"]]
```

As in serial file access, a pointer keeps track of the data item currently being accessed. Unlike serial file access, however, in random file access a specific record number within a file must be included in each random print and random read statement. The pointer is positioned at the beginning of the specified record before printing or reading occurs. Data is printed or read consecutively from the beginning of the record and cannot be read past the end of record mark.

The record number represents the location of a record in a specific file. This number can be any integer or expression which does not exceed the number of records in the file. The `rpri` statement prints data items in the form of variables, numbers, strings or substrings of characters from the beginning of the specified record. Using the `rpri` statement, each record can hold only 256 bytes (1 record) of data.

Either "end" or "ens" can be used as the last parameter in an `rpri` statement. Using "end" causes an end of file (EOF) mark to be written after the last item in the statement is printed. If "end" is not included, an end of record (EOR) mark is printed. This makes it easy to find out how much data is stored in a file using the `type` function explained later.

When "ens" (end suppress) is used as the last parameter, a list of data items is printed without printing an EOR or EOF mark. Because of this, use "ens" with care. (See the "ens" example on page 3-9.)

The program below prints consecutive numbers in each odd numbered record of a 10 record file named Ten.

```

0: open "Ten",10
1: 1→R→A
2: files Ten
3: if R>10;end
4: rpt 1;R;A;
   "end"
5: R+1→R
6: R+2→R
7: ato 3

```

In line 4, the record number is specified by the variable R. Line 6 increments this variable by 2 so that only odd numbered records are accessed.

By printing in specific records of file Ten, previous data in those records is erased and replaced by new data. An EOF marker is automatically placed at the end of each data list (i.e., the one data item A) in each odd numbered record.

File Ten now contains the following information.

Record Number	Data
1	1 EOF
2	EOF
3	2 EOF
4	EOF
5	3 EOF
6	EOF
7	4 EOF
8	EOF
9	5 EOF
10	EOF

The following program erases every third record of file Ten, which was opened and accessed previously.

```

0: 1→A
1: files Ten
2: rpt 1;A
3: A+3→A
4: if A>10;ato 6
5: ato 2
6: end

```



The information which is now left in the file is shown below.

Record Number	Data
1	EOR
2	EOF
3	2 EOF
4	EOR
5	3 EOF
6	EOF
7	EOR
8	EOF
9	5 EOF
10	EOR

The "ens" parameter can be used to write data in a record without placing any EOR or EOF marks after the item printed.

To place an EOR at the beginning of a record, execute a random print without a data list:

```
RPRT file number ; record number
```

A serial read or random read will now skip over that record and continue reading at the next record.

To place an EOF at the beginning of a record, use a random print with "end" but no data list:

```
RPRT file number ; record number ; "end"
```

An EOF causes a serial or random read to terminate at that point. As with an EOR, the data in that record is inaccessible.

## Random Read Statement

The random read (`r read`) statement reads numbers and strings from a specified record in a file, starting from the beginning of that record. A variation of this syntax can be used to reposition the file pointer.

```
r read file number » record number [ » data variables]
```

As in the case of serial read statements, the variables into which you read data items do not necessarily have to be the same variables used to print the data items in the record, but they must be the same type and in the same order.

The following program reads the data printed in the 5th and 9th records of the file named Ten.

```
0: files Ten
1: rread 1,5,X
2: rread 1,9,Y
3: prt "Data
  item 1=",X
4: prt "Data
  item 2=",Y
5: end
```



This data was originally printed in odd numbered records of file Ten. The data in records 1 and 7 was erased. The program above reads the data from records 5 and 9 and then prints it. If the calculator were programmed to read data from each record, error F0 would be displayed, indicating that an EOR mark was detected at the beginning of record 1.

The printout from this program is shown below.

```
Data item 1=
3.00
Data item 2=
5.00
```

## Positioning the Pointer

As mentioned earlier, pointers are maintained by the disc system to specify where data storage or data retrieval begins.

The pointer can be positioned at the beginning of a specified record in a file by executing a random read statement without a data list:

```
rread file number, record number
```

This positions the pointer at the beginning of the specified record. A serial read statement can then be executed, to access the beginning of the first data item of that record.

The pointer is automatically positioned at the beginning of the first record in a file after execution of a `files` or an `asen` statement. It is positioned at the next available storage location in the file after execution of a random or serial print statement.

To see how this works, first use the following program to store consecutive numbers beginning from the eleventh record of a 15 record file named Data 15.

```
0: open "Data15"
   ,15
1: files Data15
2: I+I
3: rread I,11
4: sprt I,I,"end"
5: I+1→I
6: ato 4
7: end
```

The `files` statement (line 1) sets the pointer to the beginning of the first record in the file. The pointer is repositioned to the beginning of the eleventh record of Data15 by executing line 3.

After printing in records 11 thru 15 of Data15, `error F0` is displayed. This indicates that the end of the file has been reached and no additional data can be printed in the file.

The following program is used to read the data from the beginning of record 14 .

```

0: files Data15
1: rread 1,14
2: sread 1,A,B,
  C,D,E,F,G,H
3: prt A,B,C,D,
  E,F,G,H
4: goto 2
5: end

```

The `files` statement (line 0) automatically sets the pointer to the beginning of the first record. The pointer is repositioned to the beginning of the fourteenth record in Data15 by executing line 1. The serial read statement begins reading data from that point on.

Since each full precision number uses 8 bytes of memory, 32 numbers can be printed in a (256 byte) record. On the file named Data 15 , the following numbers are stored on these corresponding records.

Record Number	Numbers
1 thru 10	none
11	1-32
12	33-64
13	65-96
14	97-128
15	129-160

The previous program reads the data on records 14 and 15 (i.e., numbers 97 thru 160) and lists this information as shown in the printout below.

```

          97.00
          98.00
          99.00
         100.00
         101.00
         102.00
         103.00
         104.00
         105.00
         106.00
         107.00
         108.00
         109.00
         110.00
         111.00
         112.00
         113.00
         114.00

```

```
115.00  
116.00  
117.00  
118.00  
119.00  
120.00  
121.00  
122.00  
123.00  
124.00  
125.00  
126.00  
127.00  
128.00  
129.00  
130.00  
131.00  
132.00  
133.00  
134.00  
135.00  
136.00  
137.00  
138.00  
139.00  
140.00  
141.00  
142.00  
143.00  
144.00  
145.00  
146.00  
147.00  
148.00  
149.00  
150.00  
151.00  
152.00  
153.00  
154.00  
155.00  
156.00  
157.00  
158.00  
159.00  
160.00
```

`error F0` is displayed at this point, indicating that an EOF mark is reached and that there is no remaining data to be read. This error message can be avoided by using the `on end` statement.

## On End Statement

The on end statement sets up a branching condition in the program. This avoids error F0, file overflow, making it possible to use a file whose exact contents are unknown.

```
on end file number ; line specifier
```

The line specifier can be either a line number (numeric expression) or a line label (text in quotes or string expression). The program branches to the specified line if:

- an EOF is encountered or the end of file is reached during a serial read.
- an attempt is made to serial print beyond the end of a file.
- an EOR or EOF is encountered during a random read or an attempt is made to random print beyond the end of file.

In the previous program, error F0 is displayed after the completion of the program, telling you that an EOF mark is encountered and no more data can be read. This error message can be avoided by including an on end statement in the program.

Here's the modified program:

```
rread 1,1;sread 1,N$
sprt 1,100,"ens"
```



When an on end condition is active and an EOF is encountered, the program branches to the specified line. The rest of the current line is not executed. This branching condition prevails until another on end is executed. All on ends are cancelled when RUN is pressed or a files statement is executed. An asgn statement only cancels the on end condition for the specified file number. Also, any on end conditions for lines overlaid by chain or get are cancelled.

Here's the preceding example program with an on end branch added:

```
0: files Data15
1: rread 1,14
2: on end 1,6
3: sread 1,A,B,
  C,D,E,F,G,H
4: prt A,B,C,D,
  E,F,G,H
5: ato 3
6: prt "End
  of File"
7: end
```

In this program, when all the data is read, the pointer comes to an EOF mark. The `on end` statement (line 2) causes the program to branch to line 6 when the read statement (line 3) encounters the EOF mark. At this point, lines 6 and 7 are executed, informing you that the EOF mark is the next item in the file.

For fastest execution, the `on end` statement need be executed only once before entering the read/print loop. This is because the program would be slowed considerably and unnecessarily if the `on end` statement were executed each time the loop is executed.

An `on end` statement sets up a condition to be executed when an EOR or EOF mark is detected to change the program flow. If you attempt to access a non-existent or invalid record without a previously executed `on end` statement, error F0 is displayed. If a `files` or `open` statement has not yet been executed, error F6 results when the `on end` statement is executed.

When using the `on end` statement, it is better to use labels since the line number parameters are not changed following program editing. Therefore, in the previous example, if line 4 were deleted, the `on end` statement would still access line 6 instead of line 5 as it should.

---

#### IMPORTANT

It's best to use labels with `on end`, since line number parameters are not changed following program editing. If line 4 were deleted in the last example, for instance, the `on end` statement would still access line 6, not the correct line.

---

## Type Function

The `type` function is used to identify the type of the next item in a specified file.

```
type ([-] file number )
```

The `type` function indicates what the next data item is by returning a value. It can be used before a read statement for this purpose. The return values are listed on the next page.

Value	Meaning
0	type undefined
1	full precision number
2	string (complete in one record)
2.1	the first part of a string (which overlaps record boundaries)
2.2	an intermediate part of string (which overlaps record boundaries)
2.3	the end part of a string (which overlaps record boundaries)
3	end of file mark or physical end of file
4	end of record mark

If `type` is executed with a positive file number, values 0 thru 3 can be returned; if `type` is executed with a negative parameter, a 4 can be returned in addition to the other values listed. A negative parameter in the `type` function is useful for detecting EOR marks when `rread` statements are used. When a positive parameter is used with serial reads, EOR marks are ignored and the type of the first item in the next record is found.

The following programs illustrate the `type` function.

```

0: open "type1",
  5
1: files type1
2: dim A[1],B#[2
  0]
3: 1111→A[1]
4: "XXXXXXXXXX"→
  B#
5: sort 1,A[1],
  B#
6: end

```

```

0: files type1;
  dim A#[1000]
1: "A":jmp int(t
  ype(-1))+1
2: prt "unknown
  type" ;stp
3: sread 1,X;
  prt "number",X;
  goto "A"
4: sread 1,A#;
  prt "String",
  A# ;goto "A"
5: prt "EOF Mark
  " ;stp
6: type(1)→X;
  prt "EOR Mark";
  goto "A"
7: end

```

A negative file number causes `type` to return the item type without moving the file pointer. A positive file number, however, causes `type` to check for EORs or the end of file. If either is found, the file pointer is moved to the start of the next record and the type of the first item is returned. (If that item is an EOF, the file pointer is advanced to the next record, and so on.)



## Data Storage Requirements

A full precision number requires eight bytes of memory for storage; this means that 32 full precision numbers can be stored in one 256 byte record. An EOR mark requires two bytes of memory for storage except when the items in the record exactly fill the record (256 bytes). When the items exactly fill the record, an EOR mark isn't needed since the actual end of the record and the end of record mark are not differentiated by the read statements. On the other hand, an EOF mark is always written, to the next record, if necessary, and requires two overhead bytes.

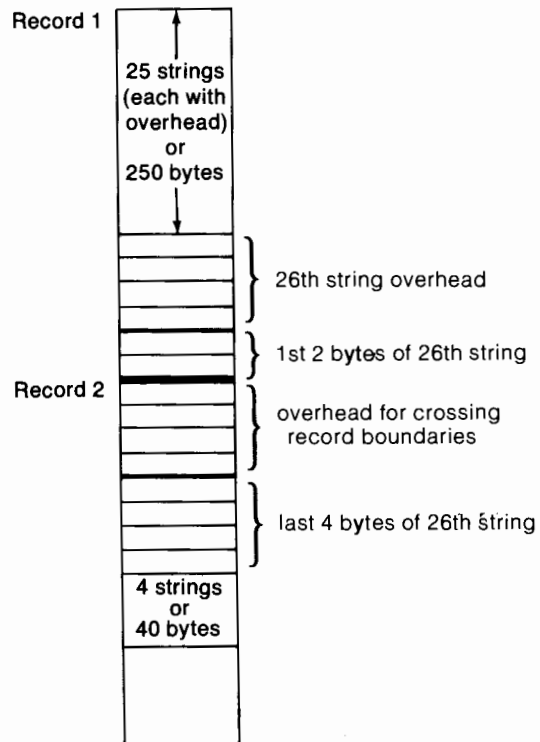
A string requires an extra four bytes of memory, called overhead, when stored, aside from the normal one byte per character storage requirement (plus one extra byte if the number of characters is odd). Therefore, up to 252 bytes of a string (plus 4 bytes of overhead) can be stored in one 256 byte record.

Strings or string arrays can overlap record boundaries using the serial print statement. Four extra overhead bytes are required for each extra record used to store the string or each string of a string array.

For example, to store a string array, A\$[30,6], each string of the array requires –

- 4 bytes of overhead (per string)
- 1 byte if the number of characters is odd
- 4 bytes of overhead whenever the string crosses record boundaries

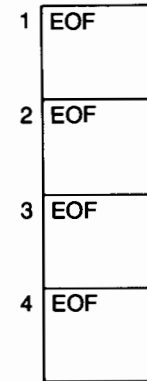
Therefore 30 strings, 6 characters long, each require 4 bytes of overhead for a total of 300 bytes, plus 4 extra bytes where the string crosses record boundaries. (If a string exactly fills a record, the overhead for crossing record boundaries is not required.)



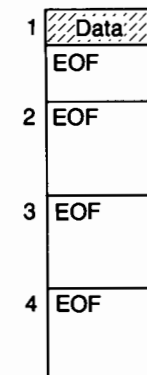
## Summary of EOR and EOF Marks

### Serial Printing

When a file is opened, EOF marks are placed at the beginning of each record. This stops any read statements but allows print statements. At the right a file with four records is opened.



Using a serial print statement, 20 numbers are stored in the first record of the file. If the "end" parameter is used, an EOF mark is printed after all data items have been stored in the file, as shown.

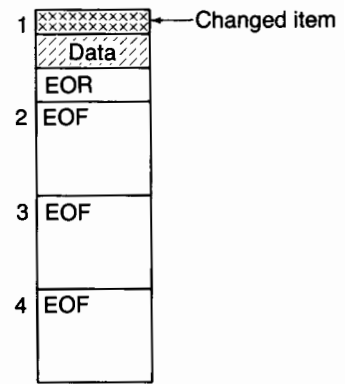


If the "end" parameter is omitted from a print statement, an EOR mark is printed after all items (20 numbers) are stored, as shown.



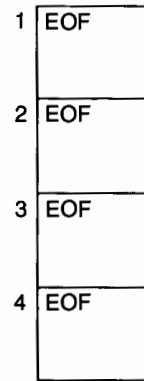
If the "end" parameter is used in a print statement (to change the first number in the data list from the previous illustration, for example), no mark is printed after the data item (or items) is stored.

The data file pointer is left where it finished when a serial print or read statement is executed.

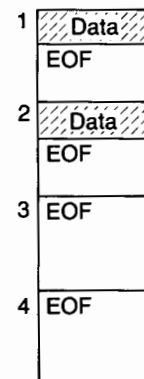


### Random Printing

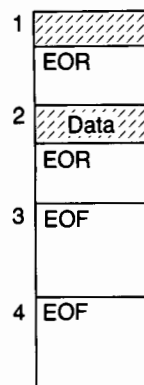
At the right a file with four records is opened and EOF marks are automatically placed at the beginning of each record.



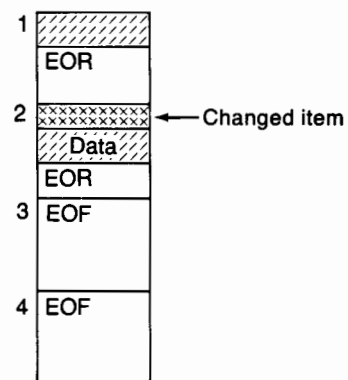
Using a random print statement, numbers are stored in the first two records of the file. If the "end" parameter is used, an EOF mark is printed after the data items in each record, as shown.



If the "end" parameter is omitted from a random print statement, an EOR mark is printed after the data items in each record, as shown.



If the "ens" parameter is used in a random print statement (to add to the first data item in the second record from the previous illustration, for example) no mark is printed after the data item is stored. Since all data in the file was not changed, the EOR mark remains unaffected. If more than 20 numbers were changed in this example, the EOR mark would be written over and not replaced leaving no mark at the end of all of the data.





# Chapter 4

## Disc Utilities

### Introduction

This chapter describes the utility statements and programs available with each disc ROM. The 98228A Disc ROM has most utilities built in, including `init` and `killall`. The 98217A ROM uses binary-language programs to run `init`, `killall`, `boot` and `vfyb`, as explained later.

These disc utilities are described:

<code>init</code>	Initializes each disc for use with the 9825 system.
<code>killall</code>	Purges all user files on a disc.
<code>boot</code>	Loads new 98217A ROM bootstraps onto a disc.
<code>vfyb</code>	Compares the 98217A ROM boots with those on the 9825/9885 disc system cartridge.
<code>copy</code>	Copies entire discs, individual files or partial data files to the same disc or another disc.
<code>repk</code>	Repacks user files on the disc.
<code>dump</code> <code>load</code> }	Transfers entire discs or disc files to and from tape cartridges.
<code>von</code> <code>voff</code> }	Sets and clears a data verification mode.

These 98217A error-recovery statements are available:

<code>tinit</code>	Initializes bad tracks.
<code>dirc</code>	Copies the spare file directory to the main.
<code>dtrk</code>	Dumps bad disc tracks to tape files.
<code>ltrk</code>	Reloads tracks dumped via <code>dtrk</code> .

## Initializing Blank Discs

Each blank disc must be initialized before it's used with your system. The 98217A Disc ROM uses the `init` utility stored on file 0 of the Disc System Cartridge. This utility is for single-sided discs in a 9885 drive. The 98228A Disc ROM has the `init` statement built in, and initializes single-sided discs with 9885 drives or single- or double-sided discs with 9885 or 9895 drives.

The initialization procedure writes addresses on the disc so that specific locations can be referenced. Test patterns are then written and read back to verify each track. The 98217A ROM also writes its bootstraps (disc control routines) immediately after initialization.

### 9885 Disc Initialization with the 98217A ROM

This procedure assumes that the 9825/9885 system is connected and powered on.

1. Insert a blank, single-sided disc in the drive. DON'T CLOSE THE DRIVE DOOR.
2. Insert the disc system cartridge in the computer.
3. Execute `trk0? ldb0`
4. Close the drive door.
5. Execute `init`
6. The first prompt is: `DRIVE NUMBER?`  
Key in the drive number (0 thru 3) and press CONTINUE.
7. The next prompt is: `DRIVE NUMBER = n . HIT STOP IF NOT .`  
`n` = the drive number you specified.  
If correct, press CONTINUE. If not, press STOP and return to step 5.

The track-by-track initialization and bootstrap loading takes about five minutes. If the utility finds six or less defective tracks, the number of defective tracks are displayed. The disc is now initialized.

If more than six tracks are found defective, the disc is rejected.

To initialize another disc, remove the first disc, insert a new one, close the door and return to step 5.

## Initializing Discs with the 98228A ROM

The init statement initializes both single-sided and double-sided discs.

```
init drive number * select code [ * interleave ]
```

The drive number is an integer expression from 0 thru 3.

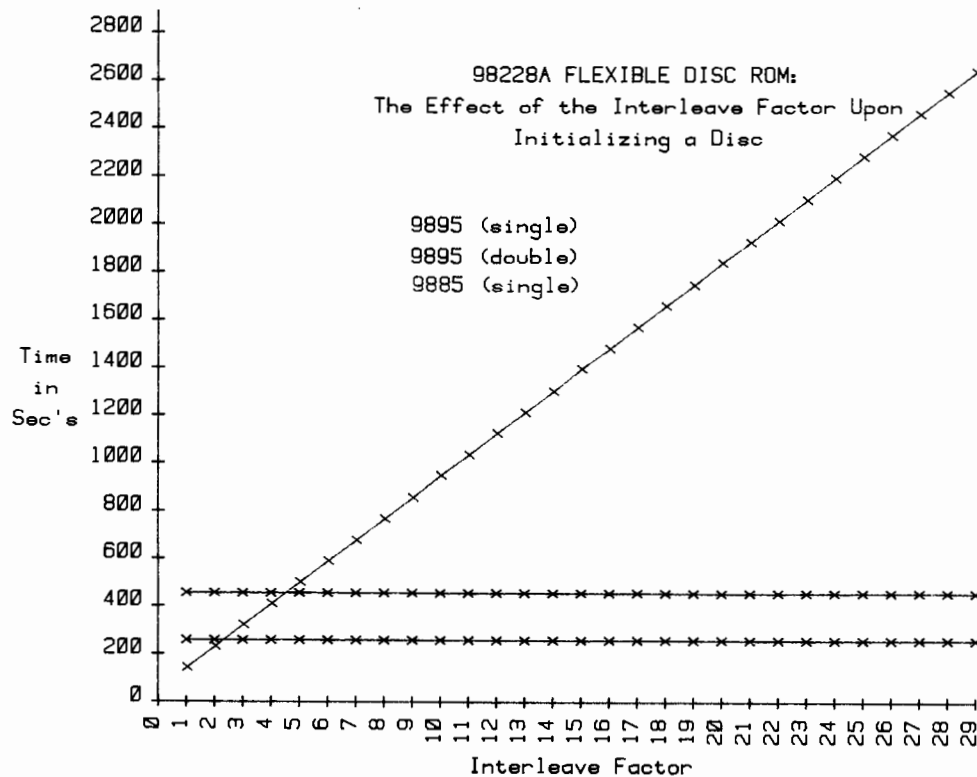
The select code can be up to a four-digit integer expression. A two-digit expression from 8 thru 15 specifies a 9885 drive via its interface select code. A three- or four-digit expression (98228A ROM only) in the form ddss specifies a 9895 HP-IB interface select code; dd is the disc controller's device address (00 thru 07), and ss is the HP interface select code (2 thru 07).

The optional interleave is an integer expression from 1 thru 29. A 9885 disc defaults to 2; a 9895 disc defaults to 7. For example:

```
init 0,8      Initializes the disc in the 9885 drive no. 0, select code 8.
```

```
init 1,102    Initializes the disc in the 9895 drive no. 1, select code 1, device address 2.
```

The initialization time depends on the disc type, the drive and the interleave factor. Typical initialization times are shown next.





## Killall Statements

The killall statement purges all user files from a disc, without affecting initialization (or the 98217A bootstraps). With the 98217A ROM, the statement must be loaded from the disc system cartridge (execute ldb 0 from either track 0 or 1). The syntax is:

```
killall
```

The disc in the default drive (0,8) is purged.

With the 98228A ROM, you must specify the drive number and select code:

```
killall drive number ; select code
```

See Addressing Disc Drives in chapter 1 for details on drive and select code parameters.

## Boot Statement

```
boot
```

The boot statement loads 98217A ROM bootstraps from the disc system cartridge to a previously initialized disc. This binary statement must be loaded from the disc system cartridge; see the list of programs on page 4-15. The bootstraps are loaded on the default (0,8) drive.

## Verify Boots Statement

```
vfyb
```

The verify boots statement compares the boots on the default drive with those on the disc system cartridge. Error B2 indicates that the boots are not the same. This binary statement must be loaded from the disc system cartridge (see the table on page 4-15).

## Repack Statement

`repk`

The repack statement moves all user files to the start of the user area on disc into a single contiguous area. This may allow creating a new file which would not previously fit into the smaller, fragmented available areas. Repacking the available user area also increases data access speed by reducing the space between user files.

A minimum of 1666 bytes must be available in read/write memory to do a repk. Error 40 indicates insufficient memory. Execute erase a to obtain the maximum available memory and execution speed.

Most calculator operations can be executed in seconds except for a long repack (where many empty spaces between files exist). This is because the directory is updated after each file is repacked. The amount of available memory also affects the repack statement: the more available memory there is, the faster repack is executed. (This is because a larger number of records can be moved at one time requiring fewer move operations.) Repack can take from a few seconds to about 15 minutes. If a power failure occurs during execution of a repack statement, one file may be lost.

If `repk` encounters a checkword (d7) or header (d5) error during its execution, it will beep and continue repacking. An unlimited number of checkword errors are allowed. One header error is allowed; if a second one occurs repacking stops and the error is displayed. Otherwise, any error(s) that occur will be displayed when `repk` is done. If more than one checkword error occurs, the last error is the one displayed.

## Verify Statements

```
von  
voff
```

The verify on (von) and verify off (voff) statements control a data-verification routine with disc print and copy statements. With verify on, each block of data written on disc is automatically read back under reduced margins and then compared with the original in memory. Use the verify routine to increase your confidence of the integrity of data just written on disc.

Data is stored on the disc in the form of binary digits represented by magnetized spots on the rotating disc. After the record is read, the checkword written at the end of the record is compared to a checkword generated by reading the record, as a check for data validity. In the verify mode, the disc is read under more stringent conditions: the time allowed to read each magnetized spot (to decode it into a 1 or a 0) is shortened. This is known as reading under reduced margins.

If your data cannot be verified, `error d9` or `error D9` is displayed. This means that your data cannot be read and must be reprinted on the disc. `error D9` indicates that data, as it appears on the disc, is not marginal and has no checkword error, but does not compare with the data just written on the disc. This may be caused by a bad interface cable connection, or a problem with the drive or calculator. If your data is marginal, `error d9` is displayed. In this case your data is readable under normal margins, but not under reduced margins. This data may not be readable for long, so it should be copied to a new area on disc or, better yet, to a new disc.

When the verify mode is disabled, the `copy` statement (explained next) can be executed in a shorter amount of time. The same is true of the print statements.

## Copy Statements

The copy statement enables you to duplicate either an entire disc, a single file or only part of a data file to either the same disc or another disc.

### Disc Copy

The disc copy statement duplicates the entire contents of a source disc onto a destination disc, record by record, including the systems area. The disc copy should be to a disc of the same type and having the same number of available tracks. If the destination disc has more tracks than the source, the extra tracks are not accessible after the disc copy since the source disc's availability tables is also copied.

```
COPY [source drive number [ ; select code ] ] "to"
      [ ; destination drive number [ ; select code ]]
```

The default drive number and select code are used when the optional parameters are omitted.

### File Copy

The file copy statement duplicates a specified file (any file type but other) to another area of the same disc or to another disc.

```
COPY source file name [ ; drive number [ ; select code ] ] ;
      destination file name [ ; drive number [ ; select code ]]
```

The default drive number and select code are used where the optional parameters are omitted.

When the source file isn't a type data file, copy automatically creates a file on the destination disc. When the source file is a data file, the destination file may or may not already exist. If it doesn't exist, copy automatically opens a new file of the same size as the source. If the destination data file already exists and is longer than the source file, an EOF is placed at the start of each extra record after data is copied.

When the destination data file already exists and is shorter than the source file, each extra record in the source file must start with an EOF. If not, error F0 indicates that the copy was aborted.

The following program shows how to copy files from one disc to another, without having to type individual file names. The program is especially useful for copying the contents of single-sided discs to double-sided discs. Since only the files are copied, not the systems area, the destination retains its original amount of user's area (not the case with disc copy). Any files already on the destination are not affected.

```

0: dim A#[35]
1: buf "cat",12500,1
2: ent "Source drive number?";D
3: ent " Source select code?";S
4: drive D,S;cat "cat"
5: for I=1 to 7;red "cat",A#;next I
6: ent "Destination drive number?";D
7: ent "Destination select code?";S
8: red "cat",A#;if len(A#);A#[1,pos(A#," ")-1]→A#;copy A#,A#,D,S;jmp 0
9: end

```

This program also shows how to use cat with an I/O buffer to read catalog information into program variables:

- line 0:            Each line of the catalog will be read into A\$ from the I/O buffer.
- line 1:            Defines an I/O buffer for a maximum-sized catalog.
- lines 2&3:        Enter the source drive number and select code.
- line 4:            Addresses the source drive and reads the catalog into the I/O buffer.
- line 5:            Skips over the first seven lines of the catalog, not needed here (see the sample catalog below).
- lines 6&7:        Enter the destination drive number and select code.
- line 8:            Reads each line of the catalog from the I/O buffer and copies the indicates file to the source disc. If the line is a null string (beyond last line of the catalog) the program exits line 8.

This technique could also be used to read other information from a disc catalog. For example, to read the drive number, select code and available number of records from this catalog:

3 blank lines {

```

DRIVE 0,8      SS
AVL RCRDS 1582
NAME TYPE SIZE      #REC TRCK RCRD
=====
grades D          100    T2    R0
Class  P   96B      1     T14   R6
Memory M 62774B    246   T5    R11
Count  P   62B      1     T13   R17
Name   P   242B     1     T13   R18
give   D          10     T13   R19
files1 P   112B     1     T13   R29
keys   K   100B     1     T14   R0
sread  P   84B      1     T14   R1
dcopy  P   266B     2     T14   R4
sprt   P   160B     1     T14   R7
sprt2  P   112B     1     T14   R8

```



use this program:

```

0: buf "buf",1000,1
1: ent "Source drive number?";D
2: ent " Source select code?";S
3: drive D;S;cat "buf"
4: fmt 1,3/,2f,/,f
5: red "buf.1";X;Y;Z

```

- line 0: Sets up an I/O buffer large enough to hold the catalog.
- lines 1 & 2: Enter drive number and select code.
- line 3: Reads the catalog into the buffer.
- line 4: Sets up an I/O format for reference by line 5.
- line 5: Reads the drive number, select code and number of a record into X, Y and Z.

For more information on cat, see chapter 1. I/O buffers and red are covered in your 9825 Extended I/O Programming manual or I/O Control Reference.

## Partial File Copy

The partial file copy duplicates a specified portion of a data file. The destination file must be assigned (asgn or files) a file number.

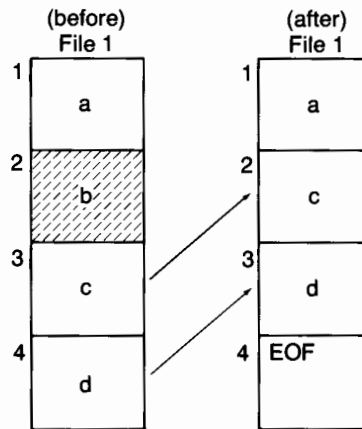
```
COPY source file number ; starting record number ;
      destination file number ; starting record number [ ; number of records]
```

After executing a partial file copy statement, the file pointers for each file involved point to the beginning of their respective files. Records before the starting points and after the ending points are ignored and unaffected.

For example, to delete a record from a file, say record 2 from a four record file, the third and fourth records are moved up to replace the second and third records and then an EOF mark is printed. This can be done by executing -

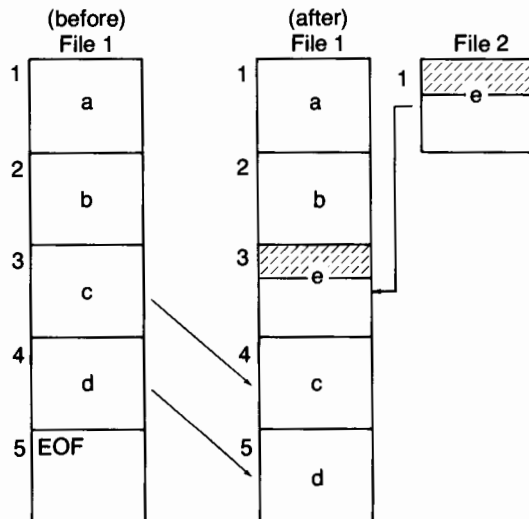
```
files File1
copy 1,3,1,2
```

This copies records 3 and 4 into records 2 and 3 and puts an EOF mark in record 4. An optional 2 can be added to the statement above to indicate that two records are to be copied. In that case, the EOF will not be written in record 4.



To add a record to a file, say a new record 3 to a five record file, you can move records 3 and 4 down to the fourth and fifth positions in the file and add a new record 3. To do this, execute -

```
filesFile1,File2
copy 1,3,1,4
copy 2,1,1,3,1
```



The first `COPY` statement copies file 1, records 3 and 4 to record 4 and 5 and checks record 5 to make sure it begins with an EOF mark. An optional 2 can be added to this statement to indicate that two records are to be copied. In that case, record 5 is not checked for an EOF mark. The second copy statement copies the first record in file 2 to the third record in file 1. The last parameter, 1, indicates that only one record is copied.

## Additional Uses for Copy

The copy statement can be used to erase the contents of a data file without purging the file. Execute:

```
copy N, 9999, N, 1
```

N is the file number, an integer from 1 thru 10.

Copy can also be used to check for an empty data file. Execute:

```
copy N, 1, N, 9999
```

## Dump Statements

The `DUMP` statements store the entire disc or indicated data files from the disc to the specified tape files.

If you execute `load` or `dump` from the keyboard, a beep is used to indicate your response to the messages displayed when you press . However beeps that occur during the actual dumping or loading process are similar to the beeps that occur during a repack operation. (See the Repack Statement.) The only difference is that any number of header errors are allowed. However, `dump` and `load` proceed only through disc errors; if a tape error occurs, `dump` or `load` stops immediately and the error is displayed.



## Disc Dump

The disc dump transfers the entire disc onto as many tape cartridges as needed (usually two thru five), starting with tape track 0, file 0.

```
dump [[-] number of records ]
```

The optional argument following the `dump` statement indicates whether the tape is to be marked automatically, or not, and the number of disc records to be dumped per tape file.

- The number of records dumped to each tape file can be 1 or 10. The default is +10. At least 2560 bytes of read/write memory are required when 10 is specified. Although no memory requirement exists when 1 is specified, the dump takes longer and requires more tape.
- A negative number of records prevents dump from automatically marking the tape before dumping records. The tape file size is 256 bytes with 1 record per tape file and 2560 bytes with 10 records per tape file.

Three tape cartridges are normally needed to dump a single-sided disc with 1 record per tape file; only two tape cartridges are needed with 10 records per tape file.


## File Dump

The file dump statement transfers the indicated data file from the disc to the specified tape file.

```
dump data file name ; tape file number [ ; [-] number of records ]
```

The optional parameter has the same effect here as for the disc `dump` statement.

The `dump` statement transfers disc information on the current track (starting at the first file number given) until the current tape track is filled (or the null file is reached, for premarked tapes). After the track is filled, if the current track is 0, `dump` automatically rewinds and continues with track 1, file 0. If the current track is track 1 `dump` displays `NEXT TAPE ; PRESS CONTINUE`, waits for tapes to be changed and then continues with track 0, file 0 of the new tape.

If the optional parameter is not given or is positive, and you insert a write protected tape, `SET RECORD ; PRESS CONTINUE` is displayed. Pull out the cartridge, slide the Record tab to the record position and reinsert the cartridge and press .

## Load Statements

The disc `load` statement transfers the entire disc from the tape files created by the disc `dump` statement.

### Disc Load

The disc `load` statement transfers the entire disc from the tape files used in the disc `dump` statement.

```
load
```

The disc is loaded starting from tape track 0, record 0. If the `dump` used ten records per tape file, there must be 2560 bytes of available memory to perform the load, or `error 40` results. With one record per tape file, there are no memory requirements.

### File Load

The file `load` statement transfers data starting from a specified tape file on the current track into the disc data file named. The tape file must have been created by a file `dump` statement.

```
load disc data file name * tape file number
```

The data file is transferred starting at the specified tape file number on the current track. The tape file number must be the same as for the corresponding `dump` statement. If the file `dump` is used ten records per tape file, there must be 2560 bytes of available memory to perform the `load`, or `error 40` results. With one record per tape file, there are no memory requirements. The size of the disc file must be greater than or equal to the size of the original file from which the data was dumped, otherwise `error DS` is displayed. Any extra records in the destination file are not affected.

## 98217A Error Recovery Routines

HPL and binary-language programs are available to recover data lost due to some programming and 9885 hardware errors. These error-recovery routines are on the 9825/9885 disc system cartridge:

Disc System Cartridge Programs

Files	Program	File Type
0 & 100	Disc Initialization (98127A)	B
1 & 101	98217A Binary Error Recovery (tinit, dtrk, ltrk & dirc)	B
2 & 102	98217A HPL Error Recovery	P
3 & 103	Checkread & Pattern Test (ckrd & ptrn tst)	B
4 & 104	HPL Disc Test (either ROM)	P
5-9, 105-109	unused	
10-69, 110-169	98217A Bootstraps	B

The checkread, pattern and HPL disc tests are described in the 9825 System Test Booklet.

Disc errors d5, d6 and d7 are recoverable. When one of these errors occurs, load the appropriate recovery routine (as explained next). DO NOT execute erase a. Also, if verify is on (von), do not execute copy, rpt or sprt.

A status message is printed immediately after the error-recovery routine is loaded. The error number, select code, track and record number are listed. If the error occurred during a data transfer, the approximate location is listed. For example:

```

error d5 was
somewhere
between
S8 D0 T9 R27
S8 D0 T11 R7

```

An "error info lost" message indicates that the error variables cannot be found; the data cannot be recovered.

## Error d5 Recovery

When error d5 occurs, data in a record can't be read because the record header is lost. To recover your data, load and run the Error Recovery routine by executing:

```
erose v  
ldbi
```

You can then read and load the contents of the track into tape cartridge files, 1 record per tape file, by first marking (30 X number of tracks involved) files, each 256 bytes in length and then executing:

```
dt r k beginning tape file number
```

Using the binary program, data can be read because the header errors are ignored. This track (or tracks) can be reinitialized by executing:

```
tinit [track number]
```

tinit without the optional number automatically initialized the track(s) in which the error occurred.

To return your data from the tape files to the reinitialized track, execute:

```
lt r k beginning tape file number
```

## Error d6 Recovery

When error d6 occurs, the calculator cannot read the data on the track because it can't recognize any of the headers on the track. The data on this track is effectively lost. When error d6 occurs, load and run the binary Error Recovery routine by executing:

```
erase v  
trkl;ldbi
```

(If possible, execute `dt rk` to dump the track, although this may not always work since the calculator may not be able to locate the track because of the loss of the headers.) Then reinitialize the track by executing:

```
tinit
```

If the track lost is track 0, the spare directory can be copied to the disc in the main directory area without affecting anything else on the disc. This can be done by executing:

```
dirc
```

The latest drive number and select code are used as the default values for the `dirc` statement.

## Error d7 Recovery (data files only)

When error d7 occurs, data in a record can't be read because the checkword for that record is not identical to the checkword generated when the record was read. To read and correct the data from the record, load and run the binary and HPL Error Recovery routine by executing:

```
ldbl
ldf2
run
```



The HPL program first asks for the track and record numbers where the error occurred. The entire contents of the record is then printed out, item by item. For each item, the item number, type number and the item itself are printed. Then the question `type?` is printed.

- If the type and the contents are correct, press . The calculator skips to the next item in the record, prints it out along with its type number and repeats the question, `type?`.
- If the type is correct, but not the contents (the item itself) enter the same type number and press . The item itself (value, string, partial string or EOR/EOF mark) is then questioned, e.g. `mantissa?` and `exponent?` or `length?`. The incorrect number, string or EOR/EOF mark can be corrected by keying the correction and pressing . As a final check, the item is reprinted and `type?` is displayed so that another correction can be made, if necessary. Then the whole procedure is repeated for the next item in the record.
- If neither the type or the contents are correct, the correct type number can be keyed in and executed. Then the item itself can be corrected. As a final check, the item is reprinted and `type?` is displayed, so that another correction can be made, if necessary. Then the whole procedure is repeated for the next item.

This is repeated for the entire record (32 numbers or the characters of a string, etc.) until the end of the record is reached. Then the calculator prints `0-back 1-ok?`. If 0 is pressed, the entire contents of the record is displayed again, item by item, to verify that all corrections were made. If 1 is pressed the record is written back to the disc.

## 98217A Binary Error Codes

- B0** Wrong syntax, argument out of range or variable not dimensioned properly.
- B1** Too many defective tracks on the disc.
- B2** Verify boots error; boots on disc don't match those on disc system cartridge.
- B3** dtrk, ltrk and tinit not allowed since error information lost or error isn't d5, d6, d7 or d9.
- B4** Attempt to access record which isn't part of a data file.
- B5** String length doesn't match length recorded in file header.
- B6** Not enough space in record buffer for data item.
- B7** Missing disc or string ROM.
- B8** Track still bad after tinit; data not recoverable.

## Appendix

# Disc Control References

## Disc ROM Syntax

The following statements and functions are available with each disc ROM except where noted. Binary utilities for the 98217A ROM must be loaded from the 9825/9885 disc system cartridge. These syntax conventions are used:

**dot matrix** - all keywords and characters in `dot matrix` must appear as shown.

[ ] - parameters appearing in brackets are optional.

**constant** - a fixed numeric value, like 2.45.

**text** - characters within quotes, like "string".

**expression** - a numeric constant (like 2), a variable (A or r3) or a numeric expression (like B+2.5).

**string expression** - text within quotes (like "hello"), a string variable name (like A\$) or a string expression like upc (A\$).

**drive number** - a numeric expression from 0 thru 3 specifying the disc drive. The drive is set to respond to the number via a switch setting.

The 98228A ROM allows a one- thru four-digit integer select code. A one- or two-digit integer (8 thru 15) specifies a 9885M drive's interface select code. A three- or four-digit number of the form ssdd specifies a 9895 drive, where ss is the HP-IB interface's select code (from 2 thru 15) and dd is the drive's HP-IB device address (from 00 thru 07). Select code 0707 is default.

**file name** - a string expression or text specifying the name of a disc file. The name can be up to six characters, but cannot contain quotes, commas, semicolons, colons, blanks, or ASCII characters less than octal 41. Each file on a disk must have a unique name. The String ROM is needed to use string expressions.

**file number** - an integer expression from 1 thru 10 specifying an assigned data file.

**line number** - a numeric expression from 0 thru 9999 specifying a program line.

**buffer name** - a string expression or text specifying the name of a data buffer (Extended I/O ROM).

**record number** - a numeric expression specifying an individual record of information (data files only).



`open file name ; file number [ ; drive number [ ; return variable ]]`

Assigns a number (1 thru 10) to an opened data file. See page 3-5. Return values are:

- |                                     |   |
|-------------------------------------|---|
| <b>0</b> File assigned.             | <b>5</b> Memory file.   |
| <b>1</b> File doesn't exist.        | <b>6</b> Binary program file.   |
| <b>2</b> Program file.              | <b>7</b> Undefined file type.   |
| <b>3</b> Special function key file. | <b>8</b> File number out of range.  |
| <b>4</b> File not defined via 9825. | <b>9</b> Data file, but logical records not<br>256 bytes long (98228A ROM<br>only). |

`boot`

A binary utility which loads 98217A bootstraps from the disc system cartridge. 98217A ROM only. See page 1-8.

`cat [ select code or "buffer name " ]`

Outputs a catalog of files on the default disc drive. Executing `cat` or `cat0` sends the catalog to the internal printer. `cat16` outputs a more-detailed catalog to the internal printer. See page 1-16. File types listed are:

- |                                     |                                 |
|-------------------------------------|---------------------------------|
| <b>B</b> Binary program file.       | <b>M</b> Memory file.           |
| <b>D</b> Data file.                 | <b>O</b> Other file (not 9825). |
| <b>K</b> Special-function key file. | <b>P</b> Program file.          |

`chain file name [ ; 1st line number [ ; 2nd line number ]]`

Loads program lines from the specified file. Same as `get` except program variables are not erased. See page 2-7.

`copy [ source drive number [ ; select code ] ; ] "to"  
[ ; destination drive number [ ; select code ]]`

Duplicates the contents of the source disc to the destination disc. Discs must be the same type, either single-sided or double-sided. See page 4-7.

`copy source file name [ ; drive number [ ; select code ]  
[ ; destination file number [ ; drive number [ ; select code ]]`

Copies a file to another disc. Omitting an address accesses the default drive. See page 4-7.

`COPY` source file number ; record number ;

destination file number ; record number ; no. of tracks

Copies the specified number of records, beginning at the indicated records.

See page 4-10.

`dir`

A binary utility which copies the spare 9885 disc directory to the main directory. 98217A ROM only. See page 4-16.

`drive` drive number [ ; select code ]

Specifies the default disc drive for successive disc operations. See page 1-14.

`dt rk` [ tape file number ]

A binary utility which dumps a bad 9885 disc track to a tape file. 98217A ROM only. See page 4-15.

`dtype`

Returns a code indicating the type of drive, disc and data format at the default disc address. 98228A ROM only. See page 1-15. Return values are:

- 0 Unable to access default disc controller.
- 1 Drive door is open or drive not present.
- 2 Drive door closed, but door was opened since last disc operation. File pointers are cleared.
- 3 9895 drive, single-sided disc, HP format.
- 4 9895 drive, double-sided disc, HP format.
- 5 9895 drive, single-sided disc, unknown format.
- 6 9895 drive, double-sided disc, unknown format.
- 7 9895 drive, single-sided disc, IBM 3740 format.
- 8 9885 drive, single-sided disc.

`dump` [[ - ] number of records ]

Dumps the entire default disc to tape cartridges. The number of records dumped per tape file can be 1 or 10. Default is 10. A negative parameter suppresses automatic marking each tape. See page 4-12.

`dump` [ file name ; tape file number ] [ ; [ - ] no. of records ]

Dumps the indicated disc file to the tape file. A negative number of records suppresses automatic tape marking before the dump. See page 4-12.

`files file name1 [ * drive number ] [ * file name2 [ * drive number ] [ * ... ]]`

Assigns names for up to 10 disc files. Specifying \* for a file name defers specification of a file name for that set of file pointers. Each file name here must be an unquoted string of text; string variables or expressions aren't allowed in files. See page 3-3.

`get file name [ * 1st line number [ * 2nd line number ]]`

Loads program lines from the specified disc file. The lines are stored, beginning either at line 0 or at the optional 1st line number. The optional 2nd line number indicates where program execution should begin. See page 2-4.

`getb file name`

Loads a binary-language program from the specified disc file. See page 2-11.

`getk file name`

Loads special function key definitions from the specified disc file. See page 2-9.

`getn file name`

Loads the specified disc memory file. See page 2-10.

`init`

A binary routine for initializing single-sided discs on the 9885 drive. 98217A ROM only. See page 4-2.

`init drive number * select code [ * interleave factor ]`

Initializes discs in either 9885 or 9895 drive. 98228A ROM only. The interleave can be an integer from 1 thru 29. See page 4-3.

`kill file name`

Purges the specified file from the default disc. See page 1-18.

`killall`

A binary program which purges all files from the default disc. 98217A ROM only. See page 1-18.

`killall drive number * select code`

Purges all user files from the specified disc. 98228A ROM only. See page 1-18.

`load`

Loads an entire disc's contents previously dumped onto tapes via dump. See page 4-13.

`load` [ file name \* tape file number ]

Loads a file previously dumped on tape to the default disc. See page 4-13.

`ltrk` [ tape file number ]

A binary routine which loads data back on to a re-initialized 9885 disc. 98217A ROM only. See page 4-15.

`on end` file number \* line specifier

Branches to the specified line when an end-of-record or end-of-file mark is read. The specifier can be a line number (numeric expression) or line label (text or string expression). See page 3-19.

`open` file name \* number of records

Creates a named data file of a specified size (256-byte records). See page 3-2.

`renn` old file name \* new file name

Renames the existing file on the default drive. See page 1-17.

`repk`

Repacks all user files on the default disc. See page 4-5.

`resolve` file name [ \* 1st line number [ \* 2nd line number ]]

Saves program lines in an existing program file. See page 2-9.

`rprt` file number \* record number [ \* variable list ] [ \* "ens" or "end" ]

Prints the list of variables into the specified data file, starting at the specified record. "ens" suppresses the EOR mark placed after data. "end" prints an EOF mark after data. See page 3-12.

`rread` file number \* record number [ \* variable list ]

Reads data from the specified data file, beginning at the indicated record, into the listed variables. Omitting the variable list just repositions the file pointer. See page 3-15.

`save` file name [ \* 1st line number [ \* 2nd line number ]]

Creates a program file and stores either the entire program (no line numbers) or only the specified lines. See page 2-2.

`savek` file name

Creates a key file and stores all special function key definitions. See page 2-9.

`sopen` file name

Creates a memory file and stores the computer's read/write memory. See page 2-10.

`spnt` file number ; variable list [ ; "ens" or "end" ]

Prints listed variables into the data file. "ens" suppresses the EOR mark printed after data. "end" prints an EOF mark after the data. See page 3-7.

`sread` file number ; variable list

Reads data from the specified file into the listed variables. See page 3-10.

`tinit` [ track number ]

A binary routine which re-initializes a bad 9885 disc track. 98217A ROM only. See page 4-15.

`type` ( [-] expression )

Returns a value indicating the next data-item type in a disc file. A positive expression causes any encountered EORs to be skipped (like with `sread`). A negative expression causes any EORs to be identified (like with `rread`). See page 3-20. Return values are:

- |   |                                   |                       |
|---|-----------------------------------|-----------------------|
| 0 | Unidentified types.               | Multi-record strings: |
| 1 | Full-precision number.            | 2.1 Start of sting.   |
| 2 | String within record.             | 2.2 Middle of string. |
| 3 | EOF mark or physical end of file. | 2.3 End of string.    |
| 4 | EOR mark.                         |                       |

`vfyb`

A binary routine which checks 98217A bootstraps on disc with those on the disc system cartridge. 98217A ROM only. See page 4-14.

`voff`

Disables data-verification with disc print and copy. See page 4-6.

`von`

Enables data-verification. See page 4-6.

## Disc Error Codes

### Hardware Errors

- d0** Firmware or driver out of synchronization. More than six defective tracks in a row. Press RESET. Too many defective tracks with 98228A init operation.
- d1** All drives in system not switched on.
- d2** Door opened while disc being accessed.
- d3** Disc not in drive or specified drive number not found. Door opened on 9895 drive.
- d4** Disc protected; writing not allowed.
- d5** Record header error. See error-recovery routines.
- d6** Track not found. See error-recovery routines.
- d7** Data checkword error. See error-recovery routines.
- d8** Hardware failure. Press RESET.
- d9** Verify error. Data is readable under normal margins but not under reduced margins. Reprint data. If problem persists, back up disc (new media) or service drive.



### Software Errors

- D0** Improper argument in disc statement.
- D1** Argument in disc statement out of range.
- D2** Improper file size; must be an integer from 1 thru 32767. No lines to store for save or savek.
- D3** Invalid file name.
- D4** Specified file not found.
- D5** Duplicate file name.  
Attempting to copy a non-data file to an existing file.
- D6** Wrong file type.
- D7** Directory overflow.
- D8** Insufficient space on disc.
- D9** Verify error. Disc controller detected no read errors, but the data read back doesn't compare with the original. Reprint data. If the problem persists, service the drive, interface or the computer.

- F0** File overflow during disc read or print.
- F1** 98217A ROM bootstraps not found. Reload boots; see chapter 4.  
Wrong memory configuration for a 98228A ROM (9825T required).
- F2** String read but wrong data type encountered.
- F3** Attempt to read data item, but type doesn't match.
- F4** Availability table overflow. Repack files on disc.
- F5** Attempt on-end branch from keyboard.
- F6** Unassigned data file pointer.
- F7** Disc is down; line cannot be reconstructed (98217A ROM only).
- F8** Disc is down and STOP pressed.
- F9** System error. Save files individually and re-initialize.

### 98217A Binary Errors

- B0** Wrong syntax, argument out of range or variable not dimensioned properly.
- B1** Too many defective tracks on the disc.
- B2** Verify boots error; boots on disc don't match those on disc system cartridge.
- B3** dtrk, ltrk and tinit not allowed since error information lost or error isn't d5, d6, d7 or d9.
- B4** Attempt to access record which isn't part of a data file.
- B5** String length doesn't match length recorded in file header.
- B6** Not enough space in record buffer for data item.
- B7** Missing disc or string ROM.
- B8** Track still bad after tinit; data not recoverable.

## Subject Index

### a

access methods (data files) ..... 1-10  
 addressing disc drives ..... 1-13  
 asgn (assign data files names) ..... 3-5

### b

backup, disc ..... 1-8  
 binary error codes ..... 4-8  
 binary program files ..... 2-11  
 boot (load 98217A ROM boots) ..... 4-4  
 bootstraps, 98217A ROM ..... 1-8,1-19,4-4  
 buffers (I/O):  
   buf with cat ..... 4-8  
   use of ..... 1-20  
 bus analyzer, use of ..... 1-20

### c

cat (file catalog) ..... 1-16  
 chain (chaining disc files) ..... 2-7  
 compatibility, data ..... 1-5  
 copy (backup) ..... 4-7  
 creating data files ..... 3-2

### d

data file operations ..... 3-1  
 data verification ..... 4-6  
 default drive ..... 1-13  
 device address ..... 1-12  
 dirc (copy 9885 disc directory) ..... 4-16

### disc:

bootstraps ..... 1-8,1-19,4-4  
 copy (backup) ..... 1-8  
 directory ..... 1-5  
 drives ..... 1-1  
 errors ..... 1-19,A-7  
 files ..... 1-9,1-16  
 ROM syntax ..... A-1  
 structure ..... 1-4  
 tests ..... 4-14  
 utilities ..... 4-1  
 double-sided discs ..... 1-3  
 drive (set default drive) ..... 1-14  
 drive number ..... 1-12  
 dtrk (dump 9885 disc track) ..... 4-16  
 dtype (identify disc type) ..... 1-15  
 dump (dump disc to tape) ..... 4-11

### e

ens (end suppress) ..... 3-7,3-9,3-12  
 end with rprr and sprt ..... 3-7,3-12  
 EOF and EOR marks ..... 1-10,3-23  
 erase data file (not purge) ..... 4-11  
 error:  
   codes ..... A-7  
   messages ..... 1-19  
   recovery ..... 1-8,4-14  
   trapping (on end) ..... 3-19

### f

files (assigning data files) ..... 3-3  
 file:  
   access methods ..... 1-10  
   assigning ..... 3-5  
   copy ..... 4-7  
   names ..... 1-12  
   numbers ..... 1-12  
   pointers ..... 3-4,3-16  
   types ..... 3-20  
   structure ..... 1-9  
 flexible discs ..... 1-3



**g**

get (load program file) ..... 2-4  
 getb (load binary file) ..... 2-11  
 getk (load special function keys) ..... 2-9  
 getm (load memory file) ..... 2-10

**h**

HP-IB programming considerations ... 1-20

**i**

IBM disc format ..... 1-5,1-15  
 index hole, disc ..... 1-3  
 init (initialize discs) ..... 1-6,4-3  
 interleave, record ..... 1-6

**k**

kill (purge files) ..... 1-18  
 killall (purge all files) ..... 1-18

**l**

linking programs ..... 2-7  
 load (tape to disc) ..... 4-13  
 load:  
   data files (rread, sread) ..... 3-10,3-15  
   key files (getk) ..... 2-9  
   memory files (getm) ..... 2-10  
   program files (get) ..... 2-4  
 ltrk (load 9885 track) ..... 4-16

**n**

non-data file operations ..... 2-1  
 numeric expression ..... 1-12

**o**

on end (trap end-of-data errors) ..... 3-19  
 open (create data files) ..... 3-2  
 other type files ..... 1-16

**p**

parallel poll with 9895 ..... 11-20  
 pointers, data file ..... 3-4,3-16  
 program files ..... 1-9, 2-1  
 purge disc files (kill, killall) ..... 1-18

**r**

record:  
   data files (rprr,sprt) ..... 3-7,3-12  
   key files (savek) ..... 2-9  
   memory files (savem) ..... 2-10  
   program files (save) ..... 2-2  
 renm (rename disc files) ..... 1-17  
 repk (repack disc files) ..... 4-5  
 resave (rerecord program files) ..... 2-9  
 return values:  
   asgn ..... 3-5  
   dtype ..... 1-15  
   type ..... 3-21  
 random file access ..... 1-11,3-12,3-15  
 record:  
   interleave ..... 1-6,4-3  
   number ..... 1-12  
 ROM memory usage ..... 1-1  
 rprr (random print) ..... 3-12  
 rread (random read) ..... 3-15

## S

save (record program file) ..... 2-2  
 savek (record keys file) ..... 2-9  
 savem (record memory file) ..... 2-10  
 sectors (tracks) ..... 1-4  
 secure programs ..... 2-11  
 select code ..... 1-12  
 serial file access ..... 1-10,3-7,3-10  
 serial poll with 9895 ..... 1-21  
 single-sided discs ..... 1-3  
 special-function key files ..... 2-9  
 sprt (serial print) ..... 3-7  
 sread (serial read) ..... 3-10  
 string expression ..... 1-12  
 storage requirements ..... 3-22  
 syntax conventions ..... 1-12

## t

tinit (initialize 9885 track) ..... 4-15  
 tracks available ..... 1-5  
 transfers (tfr) ..... 1-20  
 trapping disc errors (on end) ..... 3-19  
 type (identify data-item type) ..... 3-20

## u

utilities, disc ..... 4-1

## v

vfyb (verify disc boots) ..... 4-4  
 voff (disable data verification) ..... 4-6  
 von (enable data verification) ..... 4-6

## w

write-protect notch, disc ..... 1-3

