General Systems
Division
P. O. Box 2150
Atlanta, Georgia 30301

IBM 5110
Computing System

78-4

October, 1978

IBM 5110 NEWSLETTER

Most of this issue is dedicated to a topic we hope you will never have to use: Data Recovery. But we know, in the rush of day-to-day business, mistakes can and are made that could be at best troublesome and at worst, disastrous.

The techniques suggested here could save considerable time if you accidentally lose a file. We suggest you try them before you decide to recreate the data. And before you try these techniques, you must be certain you understand why you lost the data. If you cannot answer that question, these techniques will not help you recover the data.

Our intent is to offer alternatives to consider before you begin the process of reconstruction. At the same time, the article is presented in a theoretical approach that could help you create your own techniques for your particular situation. This is important since the techniques presented do not apply to all situations.

Also included in this issue is a procedure for using the results of SORT address out binary relative records in APL.

DATA RECOVERY: SOME SUGGESTIONS

Once in awhile, usually through accidents, recorded data becomes inaccessible or lost. This is rare, but if it happens it is annoying if not a "disaster." This article is for the user who has had this happen and sees no alternative other than to recreate the data.

Good Programming Practice

Of course the very best procedures to minimize the chances of and affects of lost data include adequate, up-to-date backup files. Some users even keep backups of the backups! Use of the write protect function can also help. (See IBM 5110 BASIC Reference Manual.) Good programming practice also suggests you build safe-guard, fail-soft, or recovery routines into your programs.

Without the "very best procedures" protecting your data, it is sometimes impossible to recover any of the "inaccessible" data, especially if the diskette itself has a flaw or is physically damaged.

Chapters 10, 11, and 12 in the IBM 5110 Customer Support Functions Reference Manual can be of great help when errors occur in file handling. Using a different drive will sometimes aid in reading data written on an area where there is physical damage.

The most probable case is when no physical error occurs, but data has become inaccessible because the wrong diskette, or the wrong file, or the wrong program is used. The result: data or the file label is destroyed. There is no recovery if all the data was destroyed.

There is hope (but no guarantee) of recovering part of the data if destruction was not complete by using the principle of substituting a good label for a bad one or by writing over unusable bytes with usable ones. Following are two examples which illustrate this principle. Application of the principle depends on your own particular situation.

Example 1: Substituting a Good Label for a Bad One

Let's assume a very large file has been inadvertently spoiled because the user OPENed to OUT with the RECL=parameter which wrote over data at the beginning of the file, and CLOSEd it. To the computer, the file appears to be only the small, most recent files but the user wants to recover as much of the original large file as possible.

To illustrate, suppose the original large file has filename "MASTER" and consists of 300 32-byte records. Let's call this the "original file":

```
RECORD-001   LARGE ORIGINAL FILE
RECORD-002   LARGE ORIGINAL FILE
RECORD-003   LARGE ORIGINAL FILE
RECORD-004   LARGE ORIGINAL FILE
RECORD-005   LARGE ORIGINAL FILE
RECORD-006   LARGE ORIGINAL FILE
RECORD-007   LARGE ORIGINAL FILE
             .
             .
             .
RECORD-292   LARGE ORIGINAL FILE
RECORD-293   LARGE ORIGINAL FILE
RECORD-294   LARGE ORIGINAL FILE
RECORD-295   LARGE ORIGINAL FILE
RECORD-296   LARGE ORIGINAL FILE
RECORD-297   LARGE ORIGINAL FILE
RECORD-298   LARGE ORIGINAL FILE
RECORD-299   LARGE ORIGINAL FILE
RECORD-300   LARGE ORIGINAL FILE
```

Unfortunately, this original file was written over by 10 22-byte records of an entirely different secondary file, also with filename "MASTER." Let's call this the "secondary file."

```
RECORD-001   SECONDARY
RECORD-002   SECONDARY
RECORD-003   SECONDARY
RECORD-004   SECONDARY
RECORD-005   SECONDARY
RECORD-006   SECONDARY
RECORD-007   SECONDARY
RECORD-008   SECONDARY
RECORD-009   SECONDARY
RECORD-010   SECONDARY
```

Conventional programming would show that the entire original file has been destroyed, because every attempt to access it gets only the secondary file. For clarity, this destroyed "inaccessible" version of the original file will be called the "spoiled" file.

Fortunately, a "programmer's trick" can recover all of those records of the original file which are contained in sectors following the sector which contains the last record of the secondary file. Here are the steps of the trick:

1 Obtain a blank "scratch" diskette with the identical VOLID as the diskette with the spoiled file. This can be verified by using LABEL DISPLAY. (See IBM 5110 Customer Support Reference Manual, SA21-9311 for this, INITIALIZATION, and COPY mentioned later.) For example:

```
        FLDNAME P LNTH    CONTENTS      FLDNAME P LNTH    CONTENTS
        LABLID 001 04 VOL1              VOLSRF 072 01 M   (2D)
NOTE →  VOLID  005 06 IBMIRD            XTNTAR 073 01
        ACCESS 011 01                   SPLREQ 074 01
        SYSID  025 13 IBM5100           PHYRLN 076 01 2
        OWNRID 038 14 OWNERID           PHYRSQ 077 02
        LBLXTN 065 01                   LBLSTD 080 01 W

        THIS IS A FORMAT8 DISKETTE
```

It can also be created by using DISKETTE INITIALIZATION.

2 Use your IMAGE COPY to copy the first diskette onto the scratch diskette. You will then have two diskettes which "look alike." Temporarily set aside the first diskette, and use the scratch diskette in steps 3, 4, 5, and 6.

3 Write a program which will write dummy records over both the secondary file and the space for the spoiled file which you just placed on the scratch diskette. The dummy records should be of the same length and number as the records of the original file. Use the same file number and filename, "MASTER." Here is an example program to do this:

```
0010 DIM A$32
0020 OPEN FILE FL1,'D40',1,'MASTER',OUT,RECL=32
0030 FOR I=1 TO 300
0040 A$='DUMMY DATA DUMMY DATA DUMMY DATA'
0050 PRINT A$
0060 WRITEFILE FL1,A$
0070 NEXT I
0080 STOP
```

4 RUN the program created in Step 3 on the scratch diskette.

5 Write a second program that will READ the entire dummy file and WRITE it into a new unused file; call it 'NEWFILE'. Mark the new file the same size as the original file. Also, this new file must later be on-line with the spoiled file. The OPEN statement for INput must precede the OPEN statement for OUTput, and a PAUSE must be the first statement following the OPEN for INput. For example:

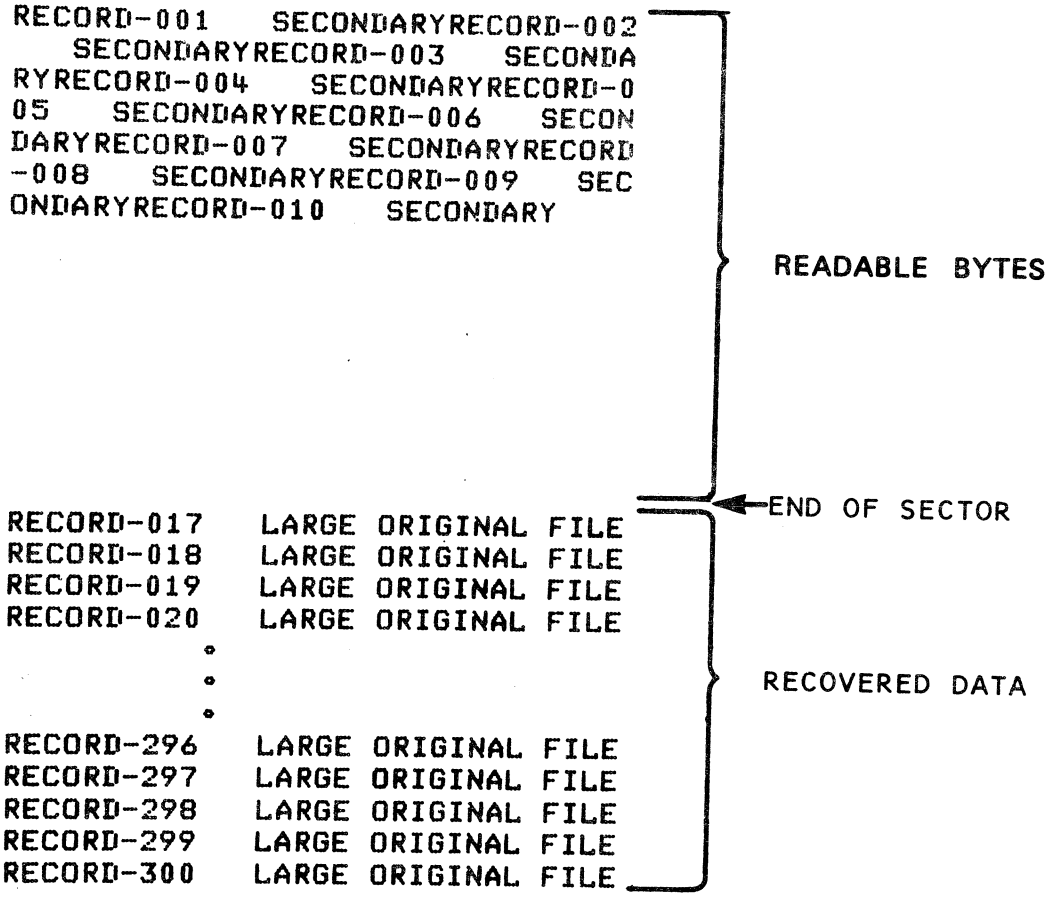
```
0010 DIM A$32
0020 OPEN FILE FL1,'D80',1,'MASTER',IN
0030 PAUSE 'SWITCH DISKETTES'
0040 OPEN FILE FL2,'D80',2,'NEWFILE',OUT,RECL=32
0050 FOR I=1 TO 300
0060 READFILE FL1,A$
0070 WRITEFILE FL2,A$
0080 NEXT I
0090 STOP
```

6 Run the program on the scratch diskette but do not press EXECUTE when PAUSE occurs.

7 During PAUSE, swap the scratch diskette for the first diskette that has the spoiled file. In effect, you are using the good label of the dummy file on the scratch diskette with the data of the spoiled file of the first diskette.

8 Now, press EXECUTE.

NEWFILE will contain all the recoverable records of the original file. You can see in the figure below that records 17 through 300 have been recovered:



Notice the important point that the spoiled file has not been altered by any of the above steps, thus preserving it in case it is needed for some other purpose.

Example 2: Writing Readable Bytes Over Unreadable Areas

In Example 1, a "programmer's trick" was used to substitute file labels in order to retrieve otherwise inaccessible records. In this example, a different trick will be used; that of making an unreadable destroyed file area disappear by writing over it. Keep in mind that you will be writing over parts of the file, thus be careful not to destroy recoverable data.

Suppose, on tape, a large stream data file (with filename "MASTER" but called the "large file") was made inaccessible by writing over it with a small data file (also with filename "MASTER" but called the "small file"). This is a greater problem for the user than the Record I/O diskette case because of a lengthy end-of-file label at the end of the small file which also destroyed data. The end-of-file label is unreadable because it does not contain the appropriate delimiters and also has destroyed additional large file data. (Note: This end-of-file label is not used on any diskette files.) In this example, we assume that the large file was not completely destroyed.

Writing over the end-of-file label with readable dummy data will be the method used to get at the recoverable large file data. Care must be used to not destroy more large file data than necessary. To illustrate, suppose the large file originally contained 400 lines of two items each, as follows:

```
  1,VARIABLE SIZE DATA
  2,VARIABLE SIZE DAT
  3,VARIA
  4,VAR
  5,VARIABLE S
  6,VARIABL
  7,VARIABLE S
          .
          .
          .
396,VARIABLE SIZE DAT
397,VARIABLE SIZE D
398,VARIABLE SIZE D
399,VARI
400,VARI
```

And these were written over by 10 items of the entirely different small file:

```
SMALLFILE
SMALLFILE
SMALLFILE
SMALLFILE
SMALLFILE
SMALLFILE
SMALLFILE
SMALLFILE
SMALLFILE
SMALLFILE
```

Conventional programming will access only this small file.

Fortunately, the use of a "programmer's trick" may recover the large file items beyond the small file end-of-file label. Here are the steps of the trick.

1 Write a program which will write dummy items over the small file. This dummy file will also have the filename "MASTER" and the same filenumber. Enough dummy items must be written to completely cover both the small file, which goes to tape as a 512-byte multiple, and its 512-byte end-of-file label. In this case, 1024 bytes must be covered. But be careful not to CLOSE because another 512-byte end-of-file label will destroy more data. Here is an example program:

```
0010 OPEN FL1,'E80',1,'MASTER',OUT
0020 FOR I=1 TO 16
0030 PUT FL1,'COVER SMALL FILE WITH  512 BYTES'
0040 NEXT I
0050 FOR I=1 TO 16
0060 PUT FL1,'COVER EOF-LABEL  WITH  512 BYTES'
0070 NEXT I
0080 PAUSE 'REMOVE TAPE'
0090 STOP
```

It is necessary to include the PAUSE to prevent the file from CLOSEing.

2 RUN the program but do <u>not</u> press EXECUTE when PAUSE occurs.

3 During PAUSE, remove the tape without CLOSEing the file because doing so would write another end-of-file label, thus destroying more data. The resulting new file will contain both the dummy records, the recoverable records of the large file, and look like this:

```
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER SMALL FILE WITH  512 BYTES
COVER EOF-LABEL  WITH  512 BYTES
COVER EOF-LABEL  WITH  512 BYTES
COVER EOF-LABEL  WITH  512 BYTES
COVER EOF-LABEL  WITH  512 BYTES
COVER EOF-LABEL  WITH  512 BYTES
COVER EOF-LABEL  WITH  512 BYTES
COVER EOF-LABEL  WITH  512 BYTES
COVER EOF-LABEL  WITH  512 BYTES
COVER EOF-LABEL  WITH  512 BYTES
COVER EOF-LABEL  WITH  512 BYTES
COVER EOF-LABEL  WITH  512 BYTES
COVER EOF-LABEL  WITH  512 BYTES
COVER E062,
VARIABLE SI'
 63,VARIABLE SIZE DA
 64,VARIABLE
 65,VARIA
 66,VARIABLE SIZE
 67,VARIABLE SIZE
 68,VARIABLE SIZE
 69,VARIABLE SIZE D
          .
          .
          .
394,VARIABLE
395,VARIABLE SIZE DAT
396,VARIABLE SIZE DAT
397,VARIABLE SIZE D
398,VARIABLE SIZE D
399,VARI
400,VARI
```

READABLE BYTES

RECOVERED DATA

4 A program can now be written to use the recovered data.

In Summary . . .

Lost data can only sometimes be recovered. When it can, the method of recovery will depend on your own individual situation. The examples shown may not be applicable to your particular problem, but understanding the principle used may help you devise your own "programmer's trick" which will apply. Care must be taken not to destroy additional data.

Recovery of lost programs presents far more problems than recovery of data, and usually cannot be accomplished. However, before giving up, don't forget that MERGE can be tried, and sometimes works for recovering some of the undamaged lines.

APL Use of SORT Address Out Binary Relative Record Numbers

The Sort utility can sort character files produced by an APL program in two ways: by an actual file sort, where the file records are physically arranged in the sorted order specified, and by an address out sort, where the master file is left unchanged but a second file is created, containing the relative record numbers of the original file in the order they would be if the file had been sorted.

The relative record numbers can be used by APL to access the original, or master, file directly if they are first put in a form usable by APL. The records as created by SORT are 4-byte binary numbers and must be converted to decimal integers before they can be used. The following procedure could be used:

1 Open the address out file for direct access use. Read the binary relative record numbers in non-translate mode:

```
⎕WA←1 ⎕SVO 2 3⍴'CTLDAT'        Offer the shared variables

CTL←'IOR file # TYPE=N'        Open the file, giving the
                                    value for the file #.
→(22≠1↑CTL)/ERROR              Check for good open. See
                                    APL Reference Manual.
CTL←0 0                        Read the first record.
                                    For TYPE=N, this is a
                                    full sector.
```

2 Reshape DAT: the record (sector) read will contain (sector size) ÷ 4 binary relative record numbers:

```
DAT←(((⍴DAT)÷4),4)⍴DAT         DAT is reshaped.
```

3 The relative record numbers can now be created. For index origin (⎕IO) of 0:

```
RLΔRECΔNO ← ¯1 + (256 * ⌽ ⍳4) +.× ⍉ ⎕AV ⍳ DAT
```

For ⎕IO of 1:

```
RLΔRECΔNO ← ¯1 + (256 * ⌽ 0,⍳3) +.× ⍉ ¯1  + ⎕AV ⍳ DAT
```

RLΔRECΔNO is the relative record number of the master record, adjusted for relative record number 0 for record 1 (APL convention). ⎕*AV* is the APL atomic vector. The spaces are not required, but have been added to enhance readibility.

4 Repeat the process for each record (sector) in the address out file. The last record may not be completely filled with binary relative record numbers, but may be padded with zeroes. You will be easily able to detect this fact because the preceding procedure will create ¯1 relative record numbers.

At this point you are finished converting the binary values to decimal integer values, and the numbers can be used to access the master file. If the master file is accessed in order of the relative record numbers created, the result will be processing the master file in sorted order.