A2-Central™

formerly
Open-Apple

November 1989
Vol. 5, No. 10

ISSN 0885-4017
newstand price: $2.50
photocopy charge per page: $0.15

A journal and exchange of Apple II discoveries

Quick and dirty database

We're getting data-base hungry here. We use commercial database programs for many chores, but we have applications where we need to be able to manipulate a large amount of data with a large number of fields under program control. In most cases, the size of the database a commercial product will handle is not the problem; the limitation we run into is the "programmability" of the database. We're interested in finding something that has the following characteristics:

- The size of the data file should not be limited by memory.
- The data files should be reasonably compact.
- The data files should be in a standard text format.
- The full ASCII character set must be supported.
- Access for daily use should be acceptably fast.
- The commands and the data format should be extensible.
- It must be able to be script-driven.

The stumbling block has been the "script driven" item; we'd like to find something like Ashton-Tate's *dBASE* program running under ProDOS or GS/OS on an Apple II. Many other things will be needed, of course; sort routines, data import/export and report generation, and possibly relational capabilities. But the initial challenge is to look at this list and see how much of it can be prototyped up "quick and dirty" so we can test our model before we commit to any large-scale development. The prototyping sounds like a job for Applesoft.

Applesoft has a few limitations that affect this list. Access speed may become a problem for a large file if we do anything labor intensive such as searching all fields of all records, or trying to sort the entire database. With even a simple program running, though, the limitations can be examined. Let's do it right now.

One problem we'll quickly encounter is that Applesoft's "INPUT" statement is a little sensitive about colons and commas. We published an "input anything" routine in September 1985, with some bug corrections in the October 1985 issue. This time, to keep the test program small and simple, we're going to modify the *data* so that the "input anything" routine isn't needed. By beginning each data item with a quotation mark, Applesoft's INPUT will accept all printable ASCII characters in the item up to the carriage return (the character normally used to indicate the end of the item) or a second quote mark, whichever is encountered first. We can live with this limitation for our testing purposes.

The experiment begins with data culled from an AppleWorks data base file of audio recordings. Each database record consists of two fields: the recording's artist and title. The data was printed to a text file and a quote mark was added to the first of each line. Two records of the data look like this:

```
"Bush, Kate
"Lionheart
"Bush, Kate
"Whole Story, The
```

Those of you who are interested in programming using text files may be familiar with the two types of organization BASIC.SYSTEM uses for those files: sequential and random access. In sequential text files, each data item is stored immediately following the preceding item. If we indicate a carriage return character with a "¬", then our two records stored sequentially look like this:

```
"Bush, Kate¬"Lionheart¬"Bush, Kate¬"Whole Story, The¬
```

Using the random access method, on the other hand, each record is given a fixed length. The fields for the record are stored sequentially, and any unused space remaining in the record is padded with something. If we use a "Δ" for padding, for example, then our two records stored in a random access file with a record length of 30 would look like this:

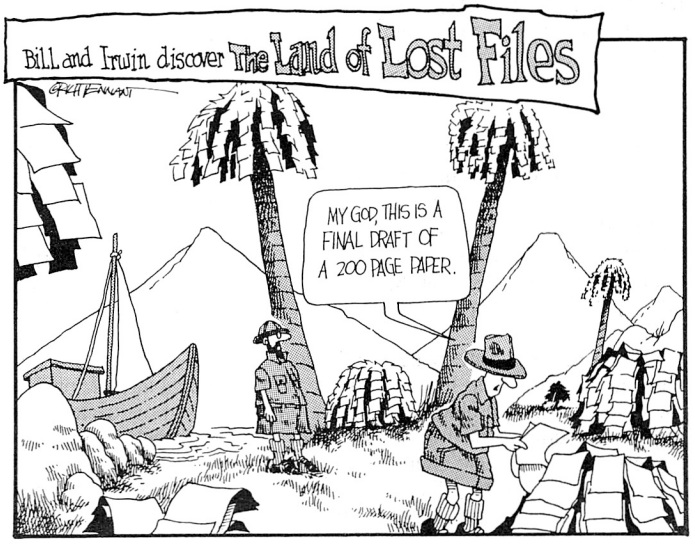
```
"Bush, Kate¬"Lionheart¬ΔΔΔΔΔ"Bush, Kate¬"Whole Story, The¬
```

(our second record is exactly thirty characters in length).

The advantage of a random access text file is that to locate record number "n", you only have to multiply the record number by the record length (assuming we call the first record in the file record "0"). BASIC.SYSTEM does this for you automatically when you specify a record number while reading from or writing to a random access file. On the other hand, by keeping the data in a sequential text file, space is saved over the use of a random access text file; every record in the random access file must be at least as long as the longest record in the file. Records that are shorter than this fixed maximum length will waste storage space.

For the 125 records in the test data used, the average record length was 29.832 bytes. The longest record was 61 bytes, and the smallest was 8 bytes. A sequential text file with the data was 3729 bytes; an equivalent random access file would be over twice as long at 7625 bytes. Our initial quest is to find a way to do basic operations (read a record, add a record, delete a record) on a disk-based data file with a compromise between acceptable performance and manageable file size.

The conventional wisdom for accessing data in the middle of a sequential text file accepts that you read and discard the preceding data. Programs such as AppleWorks manage their sequential data by loading it all into memory at once; the advantage to this method is that data in RAM can be accessed faster than data on a disk so the "read and discard" operation is quick. But this also limits the size of your database to the size of available RAM. To remain disk-based, we need a sequential text file that allows us to locate the start of any



record in the file without having to read the intervening data.

The answer is to create a second text file which will be an index file to our data. We can read the sequential text file one time and create a table of pointers that tell us at what byte offset each new record in the data file begins. We write this data into a random access text file that can be accessed quickly.

This is a simple example of a technique called ISAM; the abbreviation stands for "indexed sequential access method". By indexing the sequential text file, accessing a specific record is performed by getting the index value for the record of interest from a "table", jumping to the indexed point in the data file, and reading the record's data. Since a table of numbers pointing to positions within a file is often more uniformly sized data (varying only a few bytes in length) than information such as names or book titles, there isn't much of a penalty for using a random access file to hold the table. The limit for a ProDOS file is 16 megabytes, so a record length of 9 (8 bytes for the maximum pointer value of "16,777,215" expressed as an ASCII string, plus a carriage return) will suffice for the index file. BASIC.SYSTEM's text file commands include the ability to move to a specific byte in a file.

Working from this file model, we begin the exercise by defining a few standard variables:

```
1100  REM --- set up standard values ---
1110 D$ =  CHR$ (4): REM obligatory Control-D
1120 NFIELDS = 2: DIM F$(NFIELDS): REM two fields/record
1130 DF$ = "CDLIST": REM data file
1140 IDXF$ = "CDLIST.I": REM index file
```

NFIELDS holds the number of fields in each record; two for our test data. F$(n) will be used to hold the data for field number n. DF$ and IDXF$ are our data and index file pathnames, respectively.

Next, we want to open our workfiles and collect any additional information needed regarding them. One piece of necessary information is the end-of-file (EOF) value for the data file; when a record is added, this will become the index to the new record. There's no easy way to extract this value purely by using BASIC, so we manipulate BASIC.SYSTEM's MLI interface to pass us the EOF after we open the datafile and PEEK the value from the BASIC.SYSTEM global page:

```
1160  REM --- open each file and get EOF and record count ---
1170  PRINT D$;"OPEN ";DF$
1180  POKE 48839, PEEK (48848): REM move file ref number to parm list
1190  RESTORE : FOR I = 768 TO 778: READ HV: POKE I,HV: NEXT I
1200  DATA 169,209,32,112,190,141,10,3,24,96,0
1210  CALL 768: IF  PEEK (778) < > 0 THEN  STOP
1220 EOF =  PEEK (48840) +  PEEK (48841) * 256 +  PEEK (48842) * 65536
```

These lines create and run a small assembly language routine that gets the EOF for us.

Another necessary value to know is the number of accessible records in the datafile. This program is designed to maintain that value in record number 0 of the index file, and reads it into the variable RC at the start of the program for future reference:

```
1230  PRINT D$;"OPEN ";IDXF$;",L9"
1235  IF EOF = 0 THEN RC = 0: GOTO 1280: REM no records if "null" file
1240  PRINT  CHR$ (4);"READ ";IDXF$;",R0"
1250  INPUT "";RC
1260  PRINT D$
```

To be functional, the data base needs to perform a few simple commands. The minimum operations we determined to be necessary for experimentation were adding new data, removing obsolete data, and reading any record from the file. There are many obviously useful enhancements that could be added: sorting the data, searching the data, printing reports, editing existing data, and so on; but reading, adding, and removing records in the database file are the basic building blocks.

Our command interpreter will allow the user to "display", "add", and "remove" records. In addition, we add the capability of "packing" the data records (purging any records marked as "removed), and "quitting" the program. The format selected is simple; each command is represented by a single character ("D", "A", "R", "P", or "Q") followed by a record number for the display and remove commands. For example "D100" would display record number 100, "A" would add a new record, and "R100" would remove record 100:

```
1280  REM --- bargain basement command interpreter ---
1290  INPUT "Command?>";CMD$
1300  IF  LEN (CMD$) = 0 THEN 1390
1310 C$ =  LEFT$ (CMD$,1)
1320  IF  LEN (CMD$) > 1 THEN R =  VAL ( RIGHT$ (CMD$, LEN (CMD$) - 1))
1330  IF C$ = "A" OR C$ = "a" THEN  GOSUB 1790: GOTO 1290: REM add new record
1335  IF RC = 0 THEN  GOTO 1370: REM display, remove, pack invalid
1340  IF C$ = "D" OR C$ = "d" THEN  GOSUB 1540: GOTO 1290: REM display
1350  IF C$ = "R" OR C$ = "r" THEN  GOSUB 1540: GOTO 1290: REM remove record
1360  IF C$ = "P" OR C$ = "p" THEN  GOSUB 1920: GOTO 1160: REM pack
1370  IF C$ = "Q" OR C$ = "q" THEN  GOTO 1480: REM quit
1380  PRINT
```

In case a command is not recognized, a reminder is printed to the screen:

```
1390  PRINT  SPC( 5);"Valid commands are:"
1400  PRINT  SPC( 10);"A  - add new record"
1405  IF RC = 0 THEN  GOTO 1440: REM display, remove, pack invalid
1410  PRINT  SPC( 10);"Dn - display record #n"
1420  PRINT  SPC( 10);"Rn - remove record #n"
1430  PRINT  SPC( 10);"P  - pack database"
1440  PRINT  SPC( 10);"Q  - quit program"
1450  PRINT
1460  GOTO 1290: REM loop back for next CMD$
```

This command interface isn't very "user friendly"; for now we'll use the excuse that this is only a test. Designing a user interface is one of the more tedious parts of writing a program, and deserves more care than we're willing to commit to our experiment at this stage.

"A"dding a record requires a subroutine to accept data for the new record, write the data to the data file, add the pointer to the data to the index file, and update the record count. New data is entered from the keyboard in a loop that uses INPUT to read keyboard data into each field variable F$(n):

```
1790  REM --- handle adding record ---
1800  PRINT  SPC( 5)"Adding new record..."
1810  FOR F = 1 TO NFIELDS
1820 : PRINT  SPC( 10);"Field ";F;": ";
1830 : INPUT "";F$(F)
1840 : NEXT F
1850  PRINT
```

When the data entry is complete, a subroutine to write the data is called. When it returns, the new record is acknowledged and "A"dd returns to the command processor:

```
1860  PRINT  SPC( 5);"Please wait; writing new record..."
1870  GOSUB 2180
1880  PRINT  SPC( 10);"Record ";RC;" added."
1890  PRINT
1900  RETURN
```

The new data will be appended to the end of the file. BASIC.SYSTEM does have an APPEND command, but the EOF must be known to the program for the index values. Since EOF is always available in the program, the BASIC.SYSTEM "R"ecord and "B"yte parameters are used with READ and WRITE to move BASIC.SYSTEM's position-in-file pointer. "R0" (the first record in the file) is specified to give BASIC.SYSTEM a definite reference point, but the real offset is determined by using EOF as the "B"yte offset in line 2190. Adding the new record requires us to increase the EOF value by the length of the fields added in the loop of lines 2210 through 2240 (the math is handled in line 2230). Then the record count RC is increased by 1 and the pointer to the start of the new record is written to its place (record RC) in the index file in line 2260, and the new record count is written to record 0 (lines 2280 and 2290).

```
2180  REM --- write (append) new record ---
2190  PRINT D$;"WRITE ";DF$;",R0,B";EOF
2200 PTR = EOF: REM we'll update EOF as we go
2210  FOR F = 1 TO NFIELDS
2220 : PRINT  CHR$ (34);F$(F): REM precede with quote
2230 :EOF = EOF +  LEN (F$(F)) + 2: REM update file length
2240 : NEXT F
```

```
2250 RC = RC + 1: REM new record count
2260  PRINT D$;"WRITE ";IDXF$;",R";RC
2270  PRINT PTR: REM write record's index
2280  PRINT D$;"WRITE ";IDXF$;",R0"
2290  PRINT RC: REM write new record count
2300  PRINT D$
2310  RETURN
```

"D"isplay and "R"emove are both dependent upon a record number passed in variable R. A single routine is called to check for a valid record number (line 1550); if the record number is correct then the "D"isplay operation (lines 1600 through 1660) is performed in either case to show the contents of the record requested:

```
1540  REM --- handle display (and remove) ---
1550  IF R > 0 AND R <  = RC THEN  GOTO 1600
1560  PRINT
1570  PRINT  SPC( 5);"Sorry; record number ";R;" is invalid."
1580  PRINT  SPC( 5);"The current range is 1 to ";RC;"."
1590  GOTO 1700: REM skip display
1600  GOSUB 2080: REM get record data
1610  PRINT
1620  PRINT  SPC( 5);"Record #";R;":"
1630  FOR F = 1 TO NFIELDS
1640 : PRINT  SPC( 10);F$(F)
1650 : NEXT F
1660  PRINT
```

The display subroutine reads the index pointer for the indicated record (lines 2090 and 2100), uses the pointer in conjunction with "READ" to position itself at the start of the record in the data file (line 2110):

```
2080  REM --- read record R ---
2090  PRINT D$;"READ ";IDXF$;",R";R: REM from record R of index
2100  INPUT "";PTR: REM get pointer to record R of data
2110  PRINT D$;"READ ";DF$;",R0,B";PTR: REM point to record
2120  FOR F = 1 TO NFIELDS
2130 : INPUT F$(F): REM read data
2140 : NEXT F
2150  PRINT D$
2160  RETURN
```

Line 1670 confirms only "D"isplay was requested. If not, lines 1680 through 1690 confirm that the displayed record should be marked as "removed" and perform the exorcism by calling other subroutines:

```
1670  IF C$ = "D" OR C$ = "d" THEN  GOTO 1710
1680  PRINT  SPC( 5): INPUT "REMOVE THIS RECORD? (Y/N): ";A$
1690  IF  LEFT$ (A$,1) = "Y" OR  LEFT$ (A$,1) = "y" THEN  GOSUB 1730
1700  PRINT
1710  RETURN
```

The "R"emove subroutine displays some status information as it calls the routine to actually remove the record:

```
1730  REM --- handle remove ---
1740  PRINT  SPC( 5);"Removing record; please wait..."
1750  GOSUB 2330
1760  PRINT  SPC( 10);"Record marked as removed."
1770  RETURN
```

"Deletion" from any place other than the end of the sequential file is tricky. Our "remove" command avoids the issue by removing the pointer to the record from the list of "active" pointers in the index file, and the data is left intact in the sequential file until we do some specific house cleaning at a later time.

Unlike a random-access file, the record length within our sequential data file is not constant, and there's no guarantee that a new record written in the position of a record previously removed is going to be the same size. If the new record is smaller there is no problem; just a little "dead space" amounting to the difference of the deleted record's length in comparison to the record that replaces it. If the new record is longer, then a major catastrophe occurs: the new record will overflow the gap left by deletion of the old record and will write over the beginning of the record which follows the deleted record in the file, destroying it.

Avoiding this wholesale destruction of data is the reason for adding new records to the end of the data file, and this causes another dilemma: as records are deleted, the amount of "dead space" in the file grows. This is a drawback of the sequential organization of the data file. Still, random access files have a tougher problem if the length of a new record should happen to exceed the record length specified when the random access file was created. In the case of random access file, either the offending record has to be cut off to fit in the available space, or the contents of the file must be copied to a new file with a sufficiently larger record length (and corresponding increase in disk space used).

This program removes the reference to the record by moving the record's index pointer to the end of the index file and decreasing the number of "active" records by one, which puts the pointer "out of reach" of the commands using the "RC" value. Once the pointer is removed from the table of available records, the "removed" record (which still exists in the data file) is not accessible to the program. If the data to be deleted (and therefore its index pointer) is at the end of the file, the program skips to the routine to adjust the record count:

```
2330  REM --- delete record R (from index) ---
2340  PRINT D$;"READ ";IDXF$;",R";R
2350  INPUT "";SPTR: REM save pointer
2360  IF R = RC THEN  GOTO 2450: REM special case for last record
```

If the record is not the last record in the file, then its index pointer is saved, all the rest of the pointers are moved to "compress" the index file: and the saved pointer is written to the last record of the index file:

```
2370  FOR I = R TO RC - 1
2380 : PRINT D$;"READ ";IDXF$;",R";I + 1
2390 : INPUT "";PTR
2400 : PRINT D$;"WRITE ";IDXF$;",R";I
2410 : PRINT PTR
2420 : NEXT I
2430  PRINT D$;"WRITE ";IDXF$;",R";RC
2440  PRINT SPTR: REM save "deleted" pointer
```

Then the record count value is reduced by one and saved:

```
2450 RC = RC - 1: REM mark as deleted
2460  PRINT D$;"WRITE ";IDXF$;",R";0
2470  PRINT RC: REM update "valid" record count
2480  PRINT D$
2490  RETURN
```

Note that "A"dding a new record will cause the pointer for the last record "removed" to be overwritten and lost. If you prefer, the pointer (and even the data record it points to) could be copied to new files so that a database of "removed" records is maintained. Even in the event that the pointers are lost, a "recovery" program can be written to scan the data file to see if records exist other than those the index files indicates; if so, the recovery program can reconstruct the index pointer entry for a "hidden" record. We leave this as an exercise for the reader.

"P"acking the active records purges the "dead space" and the associated data from the files. The first thing the program does is confirm that the user really wants to do this:

```
1920  REM --- handle pack command ---
1930  PRINT
1940  PRINT  SPC( 5);"PACKING WILL PURGE ALL 'REMOVED' RECORDS!"
1950  PRINT  SPC( 5): INPUT "Is this what you want? (Y/N): ";A$
1960  IF  LEFT$ (A$,1) = "Y" OR  LEFT$ (A$,1) = "y" THEN  GOSUB 2000
1970  PRINT
1980  RETURN
```

The "P"ack subroutine maintains a safety factor by creating new workfiles and copying the "active" data from the current files into the new files. It then tries to rename the original files with names ending in ".OLD". If something goes wrong during the packing but before the renaming operation, the original data will either be available under the original filenames. If the "P"acking operation completes but the new files don't seem to work, they can be deleted and the ".OLD" files renamed to recover the original data. If the "P"acking operation is successful, the ".OLD" files need to be deleted or renamed so they don't

interfere with the next "P"acking information:

```
2000  REM --- pack prompting ---
2010  PRINT
2020  PRINT  SPC( 5);"Please wait; packing data and index files..."
2030  GOSUB 2510
2040  PRINT  SPC( 10);"Packing done. Rename '.OLD' files before"
2050  PRINT  SPC( 10);"the next packing operation."
2060  RETURN
```

To transfer the data, two new files (one for the index pointers, one for the data) ending in ".NEW" are opened:

```
2510  REM --- purge/pack data/index files ---
2520  PRINT D$;"OPEN ";DF$;".NEW"
2530  PRINT D$;"OPEN ";IDXF$;".NEW,L9"
```

Then an "update end-of-file" variable UEOF is assigned to hold the length of the new sequential data file ("0" when newly opened, of course). The index pointer for each active record is read, and the update file's index pointer (UPTR) is set equal to UEOF, the current EOF for the update file. Each record's data is transferred to the new file, UPTR is written to the new index file to mark the record's position, and UEOF is adjusted by the lengths of the record's fields.

```
2540 :UEOF = 0: REM we'll update this as we go
2550  FOR R = 1 TO RC
2560  : GOSUB 2080: REM read record R
2570 :UPTR = UEOF: REM pointer for update file
2580  : PRINT D$;"WRITE ";DF$;".NEW"
2590  : FOR F = 1 TO NFIELDS: REM write to new file
2600  :: PRINT  CHR$ (34);F$(F)
2610  ::UEOF = UEOF +  LEN (F$(F)) + 2: REM update update EOF
2620  :: NEXT F
2630  : PRINT D$;"WRITE ";IDXF$;".NEW,R";R
2640  : PRINT UPTR: REM index for record in new file
2650  : NEXT R
```

Now the record count is written to the new file:

```
2660  PRINT D$;"WRITE ";IDXF$;".NEW,R";R0
2670  PRINT RC: REM write record count
```

Finally, the original files are closed. The original files are renamed to mark them as the ".OLD" versions, and the ".NEW" files are renamed to match the original file names to become the working files. When the subroutine returns to the command processor loop (line 1360), a GOTO 1160 passes control to the initialization point of the program.

```
2680  PRINT D$;"CLOSE": REM so we can do renames
2690  PRINT D$;"RENAME ";DF$;",";DF$;".OLD"
2700  PRINT D$;"RENAME ";DF$;".NEW,";DF$
2710  PRINT D$;"RENAME ";IDXF$;",";IDXF$;".OLD"
2720  PRINT D$;"RENAME ";IDXF$;".NEW,";IDXF$
2730  REM (we'll re-open the files when we return)
2740  RETURN
```

The only remaining command is "Q"uit. All it does is close all files, say goodbye, and end:

```
1480  REM --- handle quit ---
1490  PRINT D$;"CLOSE"
1500  PRINT
1510  PRINT "See you later."
1520  PRINT D$;"BYE"
```

Finally, we have the obligatory disclaimer:

```
1000  REM === BASIC database ===
1010  REM by Dennis Doms
1020  REM published in the November 1989 issue of A2-Central
1080  REM this program is placed in the public domain
```

The program is very (no pun intended) basic. Error-checking is minimal, and recovery is near nonexistent. It's intended as a skeleton from which to build more detailed and possibly even specialized programs. The data format can be extended by using data fields to refer to records in other files (see "Making AppleWorks relational", June 1987). Commands can be added simply by adding them to the command processor along with the necessary program code.

Access speed tests were run with a larger database of 3430 records with 10 fields per record, and compared with access to the same data in a random access file. The average record length was about 99.6 bytes (the maximum was 120, the minimum 66). One hundred record numbers were generated at random, and a test program timed how long it took to position to the start of each of those 100 records on average. It averaged about 90 milliseconds for the random access file; about 170 milliseconds for the indexed file if the record indices were stored in memory, and about 210 milliseconds if they were read from the index file on disk (the disk used was a Chinook CT-30 hard drive interfaced with an Apple II SCSI card). Whether cutting the access speed in half is worth the savings in disk space is something that has to be decided for each set of real data.

If you think the design choice of the command line routine is totally undesirable, ponder this for the future: as written, the program can be directly driven with a BASIC.SYSTEM EXEC file. And there is nothing to prevent the principles from being applied to a program for another environment, in another language. It wasn't designed as a IIgs Desktop program, but there's nothing to impede that style of implementation.

ISAM has some disadvantages; the most apparent is that packing the database is a time-consuming operation. If changes to the records already in the database are infrequent, though, the need to pack the database is infrequent. We anticipate that in our applications the packing won't be necessary any more often than we need to back up the database, and the method of packing outlined in this article actually combines both operations.

By using an index file, we can even split our database into multiple files; for example, to fit on smaller disk volumes if a hard disk isn't available. Once the index value for a record is found, it can be compared to the known maximum filesize for a volume; if it exceeds that maximum, we calculate which file we need and ask the user to insert the proper disk if necessary. This also gives us a way to deal with ProDOS's 16 megabyte file size limit if our data requirements ever exceed it by modifying the index file to hold numbers larger than 16 megabytes. -DJD

AppleFest San Francisco

High points of San Francisco's September Applefest included the crowds that continue to pack Brooks Hall and the Civic Auditorium (over 19,000 people attended the show); sneak previews of some upcoming GS/OS desk accessories during the keynote speeches given by Jean-Louis Gassee and Bernard Gifford; and the first meeting of the Apple II Developers Association. Low points included Gassee publically blaming the hardware speed limitations of the Apple IIgs on the Western Design Center; the absence from the exhibit floor of the world's largest Apple II software company, Claris; and a general lack of breathtaking new products.

The new desk accessories include one that enlarges the GS/OS graphics screen two, four, or eight times for visually impared users; one that allows entry of keystrokes using only a mouse or a special headset and puff switch, for disabled users; and one that allows a VCR to be used to show short segments of video material on command, much like a laserdisk.

Claris says its marketing dollars are better spent on direct mail. This is a sensible thing for a small company to say, but it's nonsense coming from Claris. The company making most of the software sales in the Apple II third-party community has an obligation to support the Apple II. Period.

Gassee's outburst followed a question from the floor about the lack of a faster IIgs. Clearly he is tired of Apple taking the blame for this. When he pushed the blame off on the Western Design Center, however, that company's president, Bill Mensch, said the problem was that Apple had refused to work with him since the IIgs was introduced. It's impossible to figure out who's right here, but it's easy to see the deadlock between the two companies isn't doing the Apple II any good. Apple will either have to learn to work with Mensch or learn how to build its own faster 65816 microprocessor clone if we're ever going to see a faster IIgs.-TW

Ask (or tell) Uncle DOS

Corrections and amplifications

Laraine Toms, Claris Product Manager for DataFlow Computer Services in Australia, wrote to correct our statement that the AppleWorks 3.0 "no disk" upgrade will be available in Australia ("AppleWorks 3.0 upgrades", September 1989). DataFlow supports the AppleWorks 3.0 upgrade in Australia and is responsible for sending details to those Australian customers that sent checks to Claris USA along with the ***A2-Central*** *coupon as their checks are returned, but the "no disk" offer will not be valid there.-DJD*

Apple II association

The last couple of issues of ***A2-Central*** and *The Sourceror's Apprentice* have mentioned the formation of an Apple II Developer's Association.

I have seen little more in either publication other than acknowledgement of its formation. I am interested in joining such an organization. I would appreciate information on how to directly contact the organization, membership requirements, cost, and specifics about the organization's objectives and benefits.

Jerry L. Lewis
Loveland, Colo.

At the moment, there is no formal organization, no membership requirements, no dues. If you'd like to be on the group's mailing list, send your name, address, and phone number to Apple II Developer's Association, c/o Stone Edge Technologies, PO Box 3200, Maple Glen, PA 19002.

The goals for the organization are:

- *To encourage Apple Computer to promote the Apple II*
- *To promote the Apple II via the Association itself*
- *To represent the needs and concerns of Apple II developers*

The organization is intended for people making a living using the Apple II.-TW

Problems Solved

The great battery debate for the IIgs is over. The "new" IIgs machines have a clock battery in a snap-out holder.

Vern L. Mastel
Bismarck, N.D.

I was pleased to note that with System 5.0 most IIgs Desktop programs now handle the higher ASCII part of the character set automatically. This being part of the new system, with *AppleWorks GS* you need merely hold down the Option key while typing a letter and you get a different character—and it will handle accents correctly as well!

For example, in Geneva to type Mössbauer you get the "ö" by typing "Option-u" (nothing appears) then typing "o" Similar methods generate the other accents. This depends upon the font chosen, of course. There is one problem: it causes the spelling checker some difficulties! This is a vast improvement among the other improvements.

On a sadder note, it appears that *Call-A.P.P.L.E.* has shifted to a quarterly - according to my correspondence with them. This is an unfortunate step as it is one of the few worthwhile Apple II magazines available. It is even more unfortunate that this occurred without any notification. I can only suppose it is due to a lack of advertising revenue. I think this is a major loss to the Apple II world.

Stephen Harker
Oakleigh, Vic.

The IIgs System Software "Option" key handling of the alternate ASCII characters (ASCII values 128-255) is selectable from the Control Panel NDA; the Alphabet CDEV has translation options of "Standard" (the option key translation) or "None". If other programs that depend on option key sequences don't seem to recognize "option <keystroke>" with System Software 5.0, try the "None" setting.

TechAlliance's change of ***Call-A.P.P.L.E.*** *to a quarterly follows a similar change in format for their Macintosh publication (formerly there was the monthly* ***Mac Horizons****, now there's* ***Mac Tech Quarterly****). Based on the first issue of the quarterly* ***Call-A.P.P.L.E.*** *that we've seen, the information content is still there, but the quarterly format lets them print large articles without having to break them into pieces.–DJD*

Installation problems

Today I received the two ***A2-Central*** disks with the packed GS/OS System Software 5.0 files and proceeded to install it on my hard disk, which has the last version on it. When trying to install "Update System Files", I received an error of "File Not Found".

My final solution was to delete the Epson driver, and then copy the files not found into the Fonts folder. The four fonts necessary to keep the Installer going were: 1) Helvetica.10, 2) Helvetica.12, 3) Times.10, and 4) Venice.14. Apparently the Installer expects to find these files on the /System.Disk, and won't proceed without them.

Dennis Fay
Levering, Mich.

AppleWorks time display

Will some kind person write an AppleWorks 3.0 patch to display time, date, and day of week at the menus and applications like soooo many PC programs do?

Ralph Oshiro
Manhattan Beach, Calif.

John Link's ***SuperPatch 6.1*** *includes two clock patches; one specifically for the IIgs, and one for any ProDOS-compatible clock card. SuperPatch is now published by Q-Labs, Quality Computers, 15102 Charlevoix, Grosse Pointe, Mich. 48230, 313-331-0700, 1-800-443-6697; it's $39.95.-DJD*

More BASIC conversions

The BASIC Book by Harry Helms is published by McGraw-Hill Book Company, 1221 Avenue of the Americas, New York, N.Y. 10020 (ISBN 0-07-027959-4). It is a convenient guide to BASIC as used by Apple, Atari, Texas Instruments, Commodore PET, IBM, and Radio Shack personal computers.

The commands and syntax for each version of BASIC are covered, including examples in many cases. Commands, statements and functions are listed alphabetically, from ABS to XOR. There is also a handy cross-reference chart for determining in which versions of BASIC each command, statement or function appears.

Joseph Kline
Lubbock, Texas

When I received the October issue of ***A2-Central***, I saw a couple of letters that I felt I should respond to.

Page 5.66, "BASIC Conversions": I have been converting Applesoft to *GW-BASIC* for about three years and have found that for the most part Applesoft syntax will work in *GW-BASIC* as long as you don't use graphics commands and convert the screen control statements (HTAB, VTAB, etc.) to their appropriate *GW-BASIC* equivalents. The screen control and POKE statements having to do with screen control and keyboard I/O have equivalents or can be simulated in *GW-BASIC*. The POKE 36,N can be converted to LOCATE ,N+1 and the PEEK(-16384) can be simulated with the following GW-BASIC code:

```
10 A$=INKEY$: IF LEN(A$)=0 THEN 10
20 A=ASC(A$)
```

A book that Spence Earnshaw and others who are converting programs from Applesoft to other versions of BASIC might find helpful is *The BASIC Handbook* written by David A. Lien. This book covers several versions of BASIC for many popular computers, and even gives the commands for other BASICs if different. It doesn't offer a PEEKs and POKEs conversion, but if you know what they do you should be able to find an alternative method of accomplishing the same function on another machine.

Page 5.67, "BASIC editing": Gerald L. Hoffman and your other readers might also be interested in a program available on GEnie and America Online called *BPU*. This wonderful program should be in every BASIC programmer's library. I have been using this program since 1985 and would never attempt to write a BASIC program without it. This program is a pre-processor for BASIC that allows the programmer to use labels instead of line numbers and will even convert line numbers to labels. It lets the programmer pass variables to library routines as well. It is also available in CP/M and MS-DOS versions for people that also require those formats.

Page 5.71, "Evangelism beyond Apple": My condolences to John Logan and anyone else who wishes to convert to IBM. I've used Apples and IBM's for the last eight years and there's no way I'd spend my hard earned money on an IBM or compatible. Pound for pound, to me the Apple is much more powerful and a better bargain than any of the IBM's or compatibles. In eight years of sixteen hours a day use by my children (ages 2-8) as well as my wife and myself, I've spent $20 on maintenance for my Apple II Plus; in three years of use $0 on my IIgs. On the other hand, of the over 100 IBM's and compatibles where I work we have spent several thousand in just over the last two years

on machines that are used by adult professionals only. Not to mention lost time and productivity for training to use each piece of software, and still have users that feel insecure with operating the equipment. Of the users that do catch on quick and are more comfortable with the equipment, the ones that own Apples (and even Macs) far outnumber the IBM and compatible owners, most of whom only use one or two pieces of software! And there is also the number of publications for the "average user" of the Apple II compared with none for the MS-DOS based machines (I don't count the game magazines and newspapers). And even if you want to include the technical magazines, *Dr. Dobb's Journal* is the only one for MS-DOS in my opinion.

Richard Cain
Biloxi, Miss.

Joyful mouse

I have an Apple IIc and an Apple Mouse which I frequently use with *UltraMacros* for my business. I would like to add a joystick for games that my children (and their dad) are craving to own. Is there a device, similar to an A-B switch for printers, which would allow us to go from mouse to joystick without actually disconnecting any wires?

Albert M. Pagani
Toms River, N.J.

We haven't tried it, but we'd suspect an A-B switch box that used the 9-pin DB-9 style connectors (with all 9 lines switched) would work. The older Macintoshs and many MS-DOS machines use 9-pin connectors, so companies that carry accessories for these systems may have the appropriate switch box. It would probably be wise to only switch the power lines with the computer power off, though; the mouse uses some connections that the joystick doesn't.

If you would like to make your joystick act like a mouse, CH Products (1225 Stone Drive, San Marcos, CA 92069, 619-744-8546) sells a device called the ***Mirage*** *which will do the trick; it retails for $54.95.-DJD*

Perlin *Megaboard* (again)

I have the *Megaboard* by Perlin Electronics. All has worked well for two years, however Perlin does not seem to be in business anymore, at least not in the US. Does anyone know where they are?

The program that came with my card does not work properly with my IIgs. I was told by Perlin that they had a new program that did work; however, they never sent it. Can any one of your readers help?

Thomas J. Scott
Mattapan, Maine

TOPS over AppleShare

I do some computer consulting work in my spare time for local small businesses and private individuals. Although most of that work revolves around Macintosh and MS-DOS computers, a significant amount involves Apple IIs.

Since most of my clients use only a few computers, I usually recommend *TOPS* and AppleTalk when a network system is required. In my opinion it runs rings around Apple's AppleShare in terms of performance and cost, and it does not require a dedicated Macintosh server.

My problem, however, is that *TOPS* does not support the Apple II. I want to be able to connect an Apple IIgs or Apple II with a Workstation card or similar device into an AppleTalk network and not only have peripherals (LaserWriters, networked ImageWriters) but also be able to use Macintosh and MS-DOS hard drives for remote storage, transference and translation of files, etc.

I called the *TOPS* Division of Sun Microsystems and was told essentially that it was a harebrained idea and there was no money in it so *TOPS* would probably never do it. He seemed to be unaware that an Apple II would run any software other than games.

I then called Apple Corporation directly and was told no such product was anticipated. If I wanted to use an Apple II I had to buy a Macintosh and use AppleShare. Now I like Macintoshes and I would like to get one eventually, not to replace my Apple II Plus, IIe, or IIgs, but to complement them. But my clients operate on shoestring budgets, which is why (alas!) too much of my work is done gratis, and a dedicated machine is unacceptable.

Do you know of any products out there which do this? Or how do I get access to the appropriate *TOPS* protocols so I can write it myself (UGH!)? Maybe there is a patch to AppleShare which removes the requirement for a Macintosh server. I would be happy even if its operation is not as seamless as *TOPS* between the Macintoshes and MS-DOS machines.

Timothy B. Tobin
Carson, Calif.

POP, don't CALL

Can you tell me why this simple program bombs into the monitor?:

```
10 I = 1
20 PRINT I: GOSUB 30
30 I = I + 1: CALL -3288: GOTO 20
```

I'm using your suggested CALL -3288 ("Digging into DOS", ***Open-Apple***, January 1985) to clear the return stack.

Spence Earnshaw
Richmond, B.C.

CALL -3288 clears information from ONERR GOTO from the stack, but that's not the same as the return address information that a GOSUB places on the stack. Applesoft has a command specifically to remove a return address from the stack: POP. If you replace CALL -3288 with POP in your program, it will run until a floating point overflow error, Reset, Control-C, or other interruption.-DJD

Opening files

I'm confused, and do not know where to turn. It says in the *ProDOS 8 Technical Reference Manual* chapter 2.1.3 (page 14 of my copy) that you can have a maximum of eight files open at a time. I have tried to do that with some huge text files and it seems that there can be many open files but you can not read or write to them. I've tried many variations of what seems to be the logical approach i.e. OPEN file, READ file, OPEN file2, READ file2, OPEN file3, WRITE file3, etc It seems that the most files I can have open is one at a time.

How would you go about this? Is there a solution? I already looked in ***Open-Apple***...I mean ***A2-Central***, and can't find any clues. Please Help!

John Rohan
Albany, Calif.

"Open" in this context means the file has a memory buffer and a reference number assigned to it; these won't be deallocated until you close the file. The purpose of opening the file is to make information about the file available in memory (in the file's buffer) so that it won't have to be reconstructed each time you access the file. If you need to write a large amount of data to a file in several sequential operations, for example, you don't have to update a pointer to indicate where each new segment is to be written; ProDOS will track the position-in-file pointer for you as you add each segment.

Each of eight files can be open under ProDOS 8, but ProDOS can only operate on one file at a time; it must be told which file is currently being referred to and what operation (read, write, and so on) is to be performed. From machine language, this information is passed to the ProDOS machine language interface as part of the parameters for the operation. From BASIC, you use the pathname for the file along with the READ or WRITE command to tell ProDOS which file (and operation) you're interested in. A simple example would be a program to copy all strings in FILE1 to FILE2:

```
10 D$ = CHR$ (4): ONERR GOTO 90
20 PRINT D$;"OPEN FILE1"
30 PRINT D$;"OPEN FILE2"
40 PRINT D$;"READ FILE1"
50 INPUT "";A$: REM string read from FILE1
60 PRINT D$;"WRITE FILE2"
70 PRINT A$: REM string written to FILE2
80 GOTO 40: REM continue until error
90 POKE 216,0: CALL - 3288: REM clear ONERR
100 PRINT D$;"CLOSE": REM shut down
```

Both FILE1 and FILE2 are continually open in the loop from lines 40 to 80 so that ProDOS can track the current position in each file. When a file is first opened, the pointer points at the beginning of the file; each time data is read from the file, the position is updated to one byte past the end of the data just read. When a file is written to, the position is updated to one byte past the data written. So the loop reads a string from FILE1 and leaves FILE1's position-in-file pointing at the start of the next string (or the end of the file), then writes the string to FILE2 and leaves FILE2's pointer pointing at the end of FILE2. When a read is attempted at the end of FILE1, an "OUT OF DATA" error occurs and the program ends.

If the files do not remain open in the loop, the position information is lost. If FILE2 is closed, the BASIC.SYSTEM "APPEND" command can be used to move the position-in-file pointer to the end of the file before continuing to add data. But FILE1 is a bigger problem; closing FILE1 before all the data is read leaves the dilemma of knowing where the next access should resume. The program above leaves the position tracking to ProDOS; a program that closed FILE1 in the loop and wanted to resume copying strings where it left off will have to track the "position-in-file" itself.-DJD

Database date data

I think I have a solution for Keith J. Scala's date problem in the October 1989 issue. Keith was having trouble entering and sorting chronological dates in a data base of space launches.

My recommendation is to enter data in YYYYMMDD format using numbers only. For example, instead of using Sep 29 89 as the date, use 19890929 or 1989 09 29. With this method AppleWorks will not truncate 1989 or 1889 into 89.

Of course it is somewhat hard to read, but so was the conventional format when we were in first grade. Furthermore one could always enter a conventional format in another category and write a macro to do the work.

Sometimes it's best to look beyond the traditional way of expressing something.

Lee Dronick
San Diego, Calif.

GS/OS version numbers

I recently purchased GS/OS 5.0 from my dealer yet when I boot my hard disk with the space bar or escape keys pressed, I'm told I have GS/OS v3.0. Why is that?

Jon Knebel
Naperville, Ill.

The "5.0" actually refers to the IIgs System Software version. Each of the associated components such as GS/OS, the toolset files, driver files, and so on will have its own version since it logically exists as a distinct "subsystem". In the case of the version of GS/OS (and many of it's modules) supplied with System Software 5.0, the version number is 3.0.

The distinct version numbers aid in tracking the revision levels of each of the components of the IIgs System Software. The IIgs System Software is currently supplied as this "set of modules" when you upgrade your system software, but if a program needs to check a module to see if it is the proper version, it can do so.-DJD

GEnie access in Canada

I wish that GEnie would create local access numbers for those of us in Canada. I understand that they are going to link up with iNet; that will let us access GEnie via Datapac. This is supposed to happen in the fall. If and when it does can you let us know so we can join everyone else on GEnie?

Neil Butchard
Winnipeg, Manitoba

It's happened; General Electric Canada, Inc., announced the expanded service access October 3, 1989. The cost to access GEnie via the iNet 2000 gateway for 166 Canadian cities is $4.95 (Canadian) per hour billed by the local telephone companies on behalf of Telcomm Canada; this is in addition to the basic GEnie connect rates charged by GE Canada.-DJD

Odd type of files

ProDOS allows certain characters in file names: upper case letters, numerals, and periods. Lower case letters are converted to upper case. However, AppleWorks file names also have lower case letters and spaces. When I CAT an AppleWorks file in BASIC.SYSTEM with Copy II Plus, etc., as expected the spaces are converted to periods and lower case letters to upper. However, AppleWorks can then accept the illegal characters. How does AppleWorks find characters that ProDOS says don't exist in the filename?

By the way, I am very satisfied that you are Uncle DOS and not Aunt DOS. In today's liberated society, Aunt DOS would go by the name "Ms. DOS", which really wouldn't suit you.

Barry M. Austern
Cincinatti, Ohio

AppleWorks cheats. It uses the file's "auxtype" field, otherwise unused for the ADB, AWP, and ASP filetypes, to hold a "mask" to indicate the case of the filename's letters and whether a period or space should be used when the filename is displayed on the screen. The auxtype field is 16 bits long, which is enough to assign one bit to each character of the filename (with one to spare). If the bit is "0", then the character is displayed in it's ProDOS form; if the bit is "1" the character is displayed as lower case if a letter, or as a space if the character is a period. You'll notice spaces are the only "non-ProDOS" characters accepted while specifying a filename for AppleWorks use. Since AppleWorks does this translation, the filenames are displayed normally by ProDOS utilities, which generally don't use the auxtype field as AppleWorks does. In fact, Apple has reserved the use of the auxtype field for future designations of filetypes, and any use of this field must be cleared through Apple II Developer Technical support if you expect your files to work cleanly in possible future Apple II software environments. Apple II DTS now publishes ***Apple II File Type Notes*** *for recognized and documented filetypes as part of the technical note releases.*

IIgs users may notice that lower case characters are now accepted by the ProDOS FST supplied with System Software 5.0. Unlike AppleWorks, the ProDOS FST uses the ProDOS directory's version and min_version fields for each entry to store the "mask" for that entry. The case does not affect the "uniqueness" of the names; for example, you can't create files named "My.File" and "MY.FILE" in the same directory without generating an error informing you of a duplicate filename. Having lower case available does allow you to use letter cases (as opposed to just periods) to emphasize portions of the name ("LastBackup" rather than "LAST-BACKUP" or "LAST.BACKUP"). Again, most ProDOS 8 utilities won't recognize the case distinction because they aren't expecting to interpret the version and min_version information in that manner.-DJD

Pascal compatibility

In response to Shangful Nigam's question about *ORCA/Pascal* and *Apple Pascal* compatibility: while I have no experience with *ORCA/Pascal*, I have imported *Apple Pascal* programs into *TML Pascal* and have run them with little or no modification. The original structure of Pascal, if followed by the designers of the application package, should assure that what is written in UCSD Pascal should run on other strains of Pascal.

On UCSD Pascal: when I originally purchased *Apple Pascal*, I found a grand total of three books listed in *Books in Print* about *Apple Pascal*; not much to go on. While the manual that accompanies *Apple Pascal* is quite functional, it still didn't answer all of the questions that I had, and the questions that I directed towards my dealer (Apple's answers to problems) for the most part went unanswered, or if they did finally get answered, I had them figured out by then. While at a large library, I was looking through the programming language section and came across many volumes on UCSD Pascal but for the IBM. I checked out a few and lo and behold, the commands for the IBM were the same as for the Apple (about 30% of the commands were missing from the Apple version, but they are commands that one really doesn't need); there were commands that worked that Apple hadn't mentioned in their manual. The biggest point is that I leaned much from them. I guess that the message is: don't get hung up on keeping 100% within your "line". There is an awful lot of information on computers and application programs out there; information that can and does cross party lines.

Andy Conrad
Torrington, Wyo.

Assembly listing

How can I invoke the list command from a machine language program without returning and getting an error that returns me to the monitor?

Marc Kelberman
Framingham, Maine

All of the Apple II monitor ROMs have the entry point INSTDSP ($F8D0) in common; this disassembles and displays one assembly language instruction. To use the routine, you put a pointer to the address you want to disassemble into $3A (low byte) and $3B (high byte), and JSR $F8D0. When you return, $2F will contain the length of the last instruction disassembled, minus one. We incorporated this into a routine to disassemble 20 lines of code; our miniassembler program looks like this (after we add some comments):

```
0300: A9 00     LDA #00   ;point to $0300
0302: 85 3A     STA 3A
0304: A9 03     LDA #03
0306: 85 3B     STA 3B
0308: A9 14     LDA #14   ;use 20 decimal..
030A: 8D 24 03  STA 0324  ;..as our counter
030D: 20 D0 F8  JSR F8D0  ;display one instr.
0310: E6 2F     INC 2F    ;get instr. length
0312: A5 2F     LDA 2F
0314: 18        CLC       ;add to pointer
0315: 65 3A     ADC 3A
0317: 85 3A     STA 3A
0319: 90 02     BCC 031D
031B: E6 3B     INC 3B
031D: CE 24 03  DEC 0324  ;one less line to do
0320: D0 EB     BNE 030D  ;all done if 0
0322: 18        CLC       ;just for BASIC
0323: 60        RTS       ;go home
0324: 02 0B     COP 0B
0326: 00 17     BRK 17
```

The last two lines are "junk" bytes which the IIgs INSTDSP interprets as best it can. We use a memory location to hold the counter for the number of lines to disassemble because INSTDSP scrambles the contents of the A, X, and Y registers.

This same routine can be CALLed from Applesoft; just POKE the desired starting address into the routine. Or modify it to work the way you like. The CLC immediately before the RTS is there to insure the carry is clear before returning to Applesoft; otherwise, Applesoft believes a program error has been encountered. (We used CALL 768 to generate the listing above.)-DJD

Off the (Apple Desktop) Bus

I think I have found another "rotten" Apple service policy. I have had an Apple IIgs for over one and one-half years with no problems until now. I was recently writing a report and my Apple began locking up on me. I soon discovered that the left Apple Desktop Bus Port on the Keyboard wasn't working properly. This side was connected to the back of my Apple. I switch the ADB cable to the right port which works fine.

I took my keyboard to an Apple service repair shop, hoping that they would just replace the defective port for a few dollars. They told me that their policy is to trade in the keyboard for another at a cost of $99. I think this is outrageous for something so simple to repair. They also said they don't know where to get ADB ports; if they did they would replace defective ports instead.

I am writing to you, hoping that you or one of your readers knows where I could get an Apple Desktop Port for my keyboard and replace it myself. Since there are other companies that make keyboards for the IIgs, this port must be available to the public and not just Apple.

Leo G. Schnitzler
Theresa, Wisc.

My IIgs keyboard also started acting erratic, but it was the ADB connector on the right (which I use for the IIgs mouse). I opened up the keyboard and discovered that the Apple Desktop Bus connectors weren't physically connected to the supporting printed circuit board except by the "solder tails". The solder used in electronics isn't designed to be a structural material; if you flex the connector very much, it's likely it will come loose. Since there will be some pressure from trying to make sure the DIN plug is seated firmly in the socket, multiple insertion/removal cycles may cause breakage eventually.

In my case, the solder had pulled loose from the connector "pads" on the printed circuit board; if the connection was bumped, contact was broken and the IIgs started getting bad data from the mouse. Rather than trading it in, I opted to solder jumpers from the ADB connector to the proper contacts on the printed circuit board. If the connector itself is not damaged and you elect to give up any trade-in value for your keyboard, you might risk trying such a repair.

If the connector itself is bad, you may find an item from ORA Electronics, 9410 Owensmouth Avenue, PO Box 4029, Chatsworth, Calif. 91313, (818) 772-2700 to be a suitable replacement. Their order number is FDX820 ("PCB mount-female right angle 8 pin mini DIN connector"), and the connectors cost $1.60 each in lots of less than 12. We haven't actually ordered any to compare with the one on the Apple keyboard to see if they are identical; other electronics parts suppliers may stock a similar item.-DJD

A2-Central™

[redacted]

© Copyright 1989 by A2-Central

Most rights reserved. All *programs* published in ***A2-Central*** are public domain and may be copied and distributed without charge. Apple user groups and significant others may obtain permission to reprint *articles* from time to time by specific written request.

Edited by:

Dennis Doms

with help from:

Tom Weishaar	**Sally Dwyer**	**Dean Esmay**
Joyce Hammond	**Steve Kelly**	**Jay Jennings**
Tom Vanderpool	**Jean Weishaar**	

A2-Central,—titled ***Open-Apple*** through January, 1989—has been published monthly since January 1985. World-wide prices (in U.S. dollars; airmail delivery included at no additional charge): $28 for 1 year; $54 for 2 years; $78 for 3 years. All back issues are currently available for $2 each; bound, indexed editions of our first four volumes are $14.95 each. Volumes end with the January issue; an index for the prior volume is included with the February issue.

The full text of each issue of ***A2-Central*** is available on 3.5 disks, along with a selection of the best new public domain and shareware files and programs, for $84 a year (newsletter and disk combined). Single disks are $10.

Please send all correspondence to:

A2-Central
P.O. Box 11250
Overland Park, Kansas 66207 U.S.A.

A2-Central is sold in an unprotected format for your convenience. You are encouraged to make back-up archival copies or easy-to-read enlarged copies for your own use without charge. You may also copy ***A2-Central*** for distribution to others. The distribution fee is 15 cents per page per copy distributed.

WARRANTY AND LIMITATION OF LIABILITY. I warrant that most of the information in ***A2-Central*** is useful and correct, although drivel and mistakes are included from time to time, usually unintentionally. Unsatisfied subscribers may cancel their subscription at any time and receive a full refund of their last subscription payment. The unfilled portion of any paid subscription will be refunded even to satisfied subscribers upon request. MY LIABILITY FOR ERRORS AND OMISSIONS IS LIMITED TO THIS PUBLICATION'S PURCHASE PRICE. In no case shall I or my contributors be liable for any incidental or consequential damages, nor for ANY damages in excess of the fees paid by a subscriber.

ISSN 0885-4017

Printed in the U.S.A.

GEnie mail: A2-CENTRAL
Voice: 913-469-6502
Fax: 913-469-6507

Crashing success

After several months of arduous toil, faithfully serving our country's needs and fulfilling the Navy's mission, we have been rewarded with a glorious week of R&R in Bunbury, Western Australia. However, after an exhaustive and thorough search of this clean, friendly town of 25,000 inhabitants, it is patently obvious that I have no one here to talk with. Why? No Apple dealers. No Apple software. (And, in case anyone cares, no Mac's, either!)

Now for my main contribution of late, which might be of interest to some. The following is an excerpt from a letter I am sending to Apple Computer, Inc.:

"I purchased my first Apple, a IIe, in July of 1987. In January of 1988, I sold the IIe (wish that I had kept it, now) and purchased a IIgs with the following equipment and configuration:

- Applied Engineering *RAMKeeper*
- Applied Engineering *GS-RAM Plus* with one meg
- Apple Memory Expansion Board with one meg
- MDIdeas *Conserver* containing two Apple 3.5 drives
- Apple 5.25 drive
- Apple RGB color monitor

"The system is physically set up with the *Conserver* on top of the CPU, and the monitor sitting above the *Conserver*. The single 5.25 drive is daisy-chained and sits behind the rest of the equipment.

"Now, why should all of this be important to you?

"I am in the U.S. Navy, serving aboard the *U.S.S. Buchanan*. For the past four months, we have been bobbing around the Indian Ocean, North Arabian Sea, and the Persian Gulf, carrying out the Navy's mission. My system has become an indispensable aid in my work center. I use it constantly, and have two 3.5 disks containing various AppleWorks files pertinent to my work center in the two drives. (AppleWorks boots up from an 800K ROM disk supplied by the *RAMKeeper*.)

"A week ago, we were exposed to a mild storm (you probably guessed, but keep on reading), and my entire system with the exception of the keyboard **hit the deck**! It is irrelevant as to who failed to secure the restraining system; it happened. On one particularly violent roll, everything slid off the 36 inch counter top. I was informed the next morning.

"With my heart resting below the waterline, I gave the system a quick visual inspection. First damage report revealed:

"1- An inch and one-half rip in the front cover panel, upper right corner, side, of the 5.25 drive.

"2- Rubber mounting feet of 3.5 drives and RGB monitor knocked off.

"3- Height adjustment strip of RGB monitor knocked off with a 3 inch split on the left end (lower position).

"4- RGB monitor video plug loose on monitor.

"5- Internal visual inspection of the CPU case revealed no apparent damage.

"6- Although the *RAMKeeper* battery cable had become disconnected, I was informed that the transformer cable had not.

"My present rating is Interior Communications Electrician, and I am a former Electronics Technician with over 25 years total experience. So, after a thorough visual inspection, but still with hesitation, I reconnected everything and powered up. When the monitor screen remained blank, my spirits joined it! However, I was not about ready to give up totally. I powered down the system, and went back to the monitor video plug, disconnected it again, tightened up the mounting studs and reconnected the plug. Ready for the second attempt.

"Since you are reading this, it must be obvious that everything works!!!

"While I do not recommend anyone attempt to duplicate this "test" with a "messy-DOS" system, or even the "other" one Apple produces, I feel that my experience gives a loud and clear statement as to the ruggedness and quality of Apple II products. Besides an overwhelming feeling of relief after seeing AppleWorks up and ready to go on the second power-up came thankfulness for all of your engineers and quality control personnel.

"However, unless this ship capsizes, there won't be a repeat of this test on my equipment!"

Needless to say, I was very lucky. But a lot of that "luck" can be attributed to our computer and its manufacturer. The only damage sustained is purely cosmetic.

Even though I agree with the editorial comments, and others, that Apple seems to be ignoring us in favor of "Baby Brother" (*A+/inCider* editorial for June), I feel that we should also spread the word and keep encouraging those "on the inside" at Apple that we appreciate what they are doing. Also, maybe, just maybe, after my experience, the "Premium" price we are paying for (and complaining about) Apples may be justified.

I have over 25 years experience in the electronics and electronic equipment field. Accidents happen. But this is the first time that I have witnessed one of those accidents wherein the equipment (including ruggedized military equipment) sustained only minor cosmetic damage, and all parts of the system functioned normally. Try dropping your 18" portable TV three feet and see what happens! (Better yet, throw an IBM clone on the floor and see what it does!)

W. D. Watterson
San Francisco, Calif.