open apple gazette

Third Edition **Volume I, Number 3** **July/August 1982**

You are looking at the "new" OPEN APPLE GAZETTE newsletter. This is the product of the merger between the APPLE THREE newsletter and the OPEN APPLE GAZETTE. To those subscribers of the APPLE THREE thanks very much for your patience. We believe that the wait was well worth it. What you are reading is but one step in the right direction. With each issue OPEN APPLE GAZETTE will improve, in our quest to continually bring you a better publication. Our next issue will have a new look on the cover.

Due to the difficulty of merging two newsletters with different schedules it was decided to withhold publishing the second issue of APPLE THREE. APPLE THREE subscribers will receive the second issue of the OPEN APPLE GAZETTE instead. This is bound to create some confusion so an explanation is included for those subscribers.

The reason for doing this was clear: by combining the two newsletters we could pool resources and produce a better and more expanded product. An immediate benefit is that the OPEN APPLE GAZETTE will be published on a more regular basis. The other obvious improvements in quality and service are expected to benefit all OPEN APPLE GAZETTE readers. Those few readers that are duplicated in our lists will receive a refund for their APPLE THREE subscription. All APPLE THREE subscription requests that included orders for back issues: the extra funds will be applied to your 1983 subscriptions.

All readers please note: with this issue we are dedicating the new "APPLE /// NETWORK". This bulletin board is for use by Apple /// owners only. To access this board you have to dial 415-928-0412 and follow the log-on procedures. The password for the following months is TANGO. We will periodically change the password so that only ORIGINAL APPLE ///rs members will have access. Please note that the bulletin board only supports communications at 300 baud.

We are looking for user contributed software to offer to users as public domain Apple /// software. Additionally if you have programs with commercial value let us know and we will sell it to other members and pay you a royalty.

We thank you for your support and contributed articles. Keep them coming.

Original Apple ///rs

CLUB INFORMATION

MEETINGS

Meetings are held at 7:30 PM on the third Wednesday of each month. The location is the Board Room of the California Bar Association offices at 555 Franklin St. San Francisco.

MEMBERSHIP

Annual membership dues are $25 from the date application received. Your check payable to the Original Apple ///rs may be mailed to the address below.

OPEN APPLE GAZETTE POLICY

All manuscripts, photographs, and other materials are submitted free and released for publication. They become the property of the Original Apple ///rs and the Open Apple Gazette. Authors should clearly mark all material submitted for publication so that credit may be given.
The publishers/editors do not necessarily agree with, nor stand responsible for, opinions expressed or implied by other than themselves in this publication.
The Original Apple ///rs is a non-profit organization comprised of, and supported by, Apple /// owners and users. The Original Apple ///rs is run by volunteer officers and committees, and the club endeavors to aid other Apple users through this educational publication - "OPEN APPLE GAZETTE". Address all inquiries to: Original Apple ///rs, P. O. Box 813, San Francisco, CA 94101.

REPRINT POLICY

All articles appearing in the Open Apple Gazette not copywrited by the author may be reprinted by another non-profit Apple user group so long as proper credit is given to both the Open Apple Gazette and the author. Proper credit is defined as article title, author, and the words "Printed from VOL X, NO Y of the Open Apple Gazette." Permission to reprint a copywrited article may be obtained by writing to the author c/o the Original Apple ///rs.

OFFICERS

PRESIDENT 673-7635	Don Norris	(415)
VICE PRESIDENT 521-1771	Kent Hockabout	(415)
TREASURER	Julia Amaral	
SECRETARY 386-8623	Charles Coles	(415)
CONSULTANTS	Randy Fields Ken Silverman	

\- /// -

ARTICLE SUBMISSION POLICY

The Open Apple Gazette welcomes any and all articles dealing with the Apple /// Computer and its associated hardware and software. Articles may be submitted doublespaced and typewritten, or on the APPLE WRITER /// word processor.
We will send your disk back to you as soon as we output the article on our printer.

Public Domain Software for the ///

Public domain software for the Apple][was undoubtedly one of the primary reasons for its success. This software enabled owners to learn more about their machines and how to use them profitably. Public domain software for the /// has been slow in coming but here are some of the first that are available. The Applecon program from Apple Computer Inc. will greatly add to the library of public domain software for the ///. You can help with this by sending us programs you have converted to or written for the ///.

Applecon from Apple Computer Inc.

Applecon is a new utility for the Apple /// which converts Applesoft BASIC programs to Apple /// Business BASIC programs to the extent that they can be machine converted. This program will not convert any copy protected programs or diskettes. This utility will take an Applesoft (Apple II) program and move it up to SOS and into Apple Business BASIC and then will make the proper changes. Those lines it cannot convert directly

into Business BASIC will be flagged into a REM statement for you to correct. The disk comes with several pages of documentation on the disk in a text file. The file can be read by Apple Writer ///, or you can output it via the Pascal System.

File Cabinet ///

This is a small general purpose data base management system written in Business BASIC. The use of File Cabinet /// is simple and most of it is self documenting. File Cabinet provides a means of interactively defining data files, entering data, sorting, retrieving records containing specific data, deleting records, and printing reports. Because all of the data in File Cabinet is memory resident the size of the data base is limited to a relatively small amount but the handling of this data is very fast.

DOS to SOS text File converter.

This program enables you to move DOS 3.3 text files to SOS. It is useful in moving VisiCalc Models from the][to the ///. If you own Apple Writer the Apple Writer Utility diskette already will do this for you.

These diskettes are available to members for $8.50 each. Non Members $10.00. Canadian Residents add $ 1.00 for postage, add $2.00 for other foreign postage. Make your checks payable to the:

Original Apple ///rs
P. O. Box 813
San Francisco, CA
94101

- /// -

CP/M On An Apple ///

By: William C. Jacobson

The CP/M Softcard marketed by Apple has finally arrived, permitting owners of the Apple /// access to the vast amount of software available for this business oriented operating system. As touted by Apple, Owners of the /// now can use DOS, SOS, and CP/M on their marvelous machines.

I purchased one of the first copies of the Apple Softcard System available in the Washington, DC area, and would like to relate some of my initial experiences with it.

While it will be sometime before I can classify myself as an expert in matters CP/M, I have been able to wend my way through its special command structure with a minimum of problems.

One of the first questions I had about my new Softcard was the existence of any unique features that distinguish it from its many relatives. As far as I can see, there are no special CP/M related functions of significance. It simply allows me to access CP/M.

Having said this, however, I must qualify my comment in two respects:

> The SOSXFER function of the Softcard does allow transfer of ASCII files from SOS (the resident Apple /// operating system) to CP/M; and
>
> It is as yet unclear what applications programs (off the shelf CP/M software) will work on the Apple ///.

The SOSXFER function continues the Apple "tradition" with the /// of permitting easy text file interchange between operating systems. An important use of this function involves the WordStar program that I used to write this article. I have been able to quickly transfer Apple Writer /// files to CP/M for reformatting and modification. This compatibility also extends to the SpellStar spelling check software for WordStar. I am now able to prepare ASCII files using whatever word processing software seems appropriate for the file being created, and then use the special features of WordStar and SpellStar to full advantage. This statement applies both to Apple Writer and the powerful Pascal text editor available for the ///. I assume that it would also apply to Apple II files transferred with the Apple Writer /// utility disk, but I have not attempted it.

The significance of SOSXFER is easily illustrated. My teenage son is a writer

and finds Apple Writer /// an excellent program for composing articles. It is very easy to use, so that he can concentrate on what he is writing, and not on the peculiarities of the software he is using. However, he also likes the formatting features of WordStar and the spelling checks available with SpellStar. With SOSXFER he can have all the advantages of each.

There are still many questions about what applications software will or will not work on the Apple ///, without major adaptation. This article is testimony to the fact that the Apple II version of WordStar can be adapted, if you follow the special instructions listed in the Apple Technical Note on this subject (Softcard /// dated August 16, 1982).

While I intend to make extensive use of the M-BASIC software that comes with the Softcard, my primary interest is off the shelf programs. Of particular interest is the dBASE II data base management system. I have sent a letter to Ashton-Tate, the creators of dBASE, and hope to receive a favorable response in the very near future.

If some adaptation is required, the key is instructions needed to make the conversion. The Dynamic Debugging Tool (DDT) feature of CP/M allows you to make changes very easily, once it is clear what you have to do. For WordStar, it took me awhile to learn to use DDT, but the actual changes only took a few minutes. I hope that Apple will provide Technical Notes for all popular CP/M software, so that this may be a simple, uncomplicated process.

///

A Funny Thing Happened On The Way To The Perfect Program - Version 1.0

By: David D. Meisel

As a programmer with nearly 20 years of experience (I started "hacking" as a graduate student working toward a Ph.D in astrophysics when IBM 650's and 1620's were considered "state-of-the-art" and have been a slave of mainframes ever since !!) the advantages of the S.O.S. that runs the APPLE /// were crystal clear. No more worry about assembly coding, no more worry about where to put the data and program instructions and no longer would figuring out the I/O seem like working your way out of a bear pit. At last, one could concentrate on programming the problem that originally forced you to buy the personal computer in the first place. Although I had programmed previously in ALGOL, FORTRAN, and FOCAL (one of the original high level interpreter competitors of BASIC developed for DEC PDP-8 minicomputers), I decided to be lazy and try BUSINESS BASIC for starters. If the program worked then it would be easy to put it into a compiled version using APPLE /// Pascal.

Conventional wisdom says learn the language through a tutorial style book and then tackle the "official" manual, but figuring I knew all the beginning stuff I decided to get right down to the reference manual. I went through the manual and put index tabs at all the appropriate spots so that I could find the rules concerning each construct as I need them. (All reference manuals that are really useful should be constructed this way but few if any are these days...APPLE take note.) These tabs will take a lot of beating under most circumstances so use ones that are durable. Now armed with a "quick draw" version of the "bible" I was able to get through most of the easier stuff in no time at all.

Now, the APPLE /// manuals are orders of magnitude better than their mainframe counterparts in virtually every way so both the Owner's manual and the device driver manual looked straight forward and with proper index tabs were found to make things pretty easy for an old hacker like me. While some of my friends (who after months of ownership of other brands) have still not been weaned from their owner's manuals, I -- the APPLE /// guy--- was whizzing right along in his programming. So far so good.

Then I found my first snag. Breezing right along I started to look for a way to chain a whole series of programs together. On p.28 of the manual I discovered the EXEC command. This great little feature lets you chain by setting

up a text file of commands and then EXEC the text file in immediate mode. The thing that you are not told (until p.31) is that the programs so referenced cannot contain any INPUT or GET statements that need stuff from the keyboard. Both of the appropriate remedies are given on p.32 but it would have been helpful if these caveates had at least been flagged on p.28. For example, in the introductory paragraph it would have been nice to have seen something like ".....by allowing applications that do not require ANY direct keyboard entry to be run in sequential order. If keyboard entry is required then an override feature is available,but must be programmed into each program itself.(see notes on p.32)." Score APPLE /// one; Meisel zero.

Snag number two came somewhat later. As an old FOCAL programmer, I was really used to cramming as much on a line as I could. After all line numbers do cost memory space; it says so right on p.243. Under the description about STATEMENTS on p.47 it says ".....that a list of statements may share one line number, adjacent statements must be separated by a colon (:)." Fair enough, I thought, so I went blindly on my merry way. At one point in my program, I needed to be able,on option, to switch out and do a subroutine and then return to the middle of the line. That seemed easy.

```
10 IF A$="right"THEN GOSUB 80:A=B+C:IF
BB$="wrong" THEN 70:B=C©2
```

The subroutine begins at line 80 and has a RETURN in it. Straight forward application of the statement in this case will be wrong because in normal BASIC without concatenation (i.e. statement chaining on one line) the result of the first IF statement does the subroutine if A$="right" and then goes to the next line regardless of what comes next on the line containing the IF statement. When it does not work properly, you turn to p.107 and in the first paragraph you find the statement that indicates your problem - - - - A false expression in the IF forces a jump to the next line. O.K. ?????? Well not quite. Take a gander at the examples on p.107. Opps !!!!

```
")IF S/4>=17*NOT 2 THEN GOSUB
3000:INVERSE:PRINT "HI" . . . . . .
etc."
```

Thus instead of the machine doing only the subroutine 3000 when the IF clause is true it goes on and does the remaining statements in the list as well. This is just the opposite to the logic I was applying.

Well, now the question comes up of how to do what I originally intended, namely how does one get an optional switchout and return to do the rest of the line when the IF clause is false ? A partial answer is given on the next page where the ELSE construction is introduced and on p. 185 where the syntax references are given for IF... constructions. The examples given in the manual are very poor, because it is not clear from them whether the FALSE condition in the IF continues execution of the other statements in the line or not. Furthermore, the insistence that the ":ELSE" is optional (p.108 and p.186) when in fact for proper operation of the compound IF...THEN statements it is ABSOLUTELY ESSENTIAL ----- just further aggravates the situation.

In the hope that it will save time and effort for others, I offer the following example of proper use of the compound IF...THEN..:ELSE statement.

```
 5 INPUT A$
10 IF A$="TRUE" THEN A=B+C+D:
   PRINT A:GOTO 20:ELSE A=B-C-D:
   PRINT A:GOTO 30
20 PRINT B,C,D:GOTO 40
30 PRINT D,C,Ð
40 END
```

In this example when A$="true" then it computes A, prints A,and then goes down to print B,C, and D in that order. This means that if the statement between the IF and THEN is true everything between the "THEN" and the ":ELSE" is done. (Note that the GOTO 20 is not needed because 20 is the next line number, but I have included it anyway.) IF A$ is anything else other than the word "true" then the statements between the ":ELSE"

and the end of the line are performed including the printing of B,C,and D in reverse order.

The logic of the construction is perfectly sound, it is the explanation of its proper syntax that is poor. I think part of the problem is the fact that if it really were intended for the construct to indicate the nature of the operation to be that as indicated in my example, it is not really logical that the colon occur as the delimiter of choice in the "else" part. To me the construct "<space> ELSE" would have been a better choice, but of course this would not allow the ELSE to have functioned like a REM statement when the colon is omitted.(see p.108) and ELSE is used before or independently of the IF...THEN.

It is, of course, unrealistic to expect manual writers to catch everything that can be misinterpreted in their text, but it seems a pity that a little sloppy work in a part of what otherwise would be an excellent text cannot be corrected before reaching the consumer. Certainly the APPLE /// PASCAL manual has been properly updated and revised, so why not the Business Basic manual too!!!!! Until then be especially careful with your IF..THEN..ELSE uses and test each carefully before assuming they work properly.

- /// -

Apple /// COBOL

Apple announced that come this Fall you will be able to buy COBOL for the Apple ///. According to Apple Computer, Apple /// COBOL has been certified by the General Services Administration's (GSA) Federal Compiler Testing Center at high-intermediate level, which is a higher level than many of the COBOL systems available today for minicomputers.

One feature of Apple /// COBOL stands out: Animator, a powerful screen-oriented, source-level debugger. Animator allows the programmer to run a program one statement at a time or continuously while watching its execution which could give it applications in a teaching system. Animator provides an "animated" view of actual program execution, and it can stop program execution at any time to allow for checking and changing of data items. A truly useful method of program debugging. The full use of the Animator requires 256K of memory.

Another feature is FORMS-2, a COBOL source-code generator which lets the programmer begin with a blank screen and end with a fully operational program. FORMS-2 interactively creates data entry screens and generates COBOL source for use in a program. Price will be under $500.

Pascal File Access Program

Record Processing Services (RPS) is a sophisticated, multi-keyed file access program that saves Apple /// Pascal software developers costly development time. RPS provides file management services for programs handling large quantities of data. Programs built on RPS can access each others' files for data manipulation or interchange. Apple is committed to RPS as its own Apple /// access standard. RPS has the following specifications:

- --Maximum file size is 16 megabytes
- --Has 6 access modes, 11 data types
- --Permits up to 8 keys per file and multi-field keys
- --Supports variable and fixed length
- --Utilizes B-Tree index structure
- --Can scan multiple files simultaneously
- --Supports ProFile hard disk

RPS also provides 64 bit integer numerics that are primarily designed for accounting and business programs. This data type allows users to work with numbers as large as +/- 9,999,999,999,999,999,999 (9 quintillion?) a feature which prepares it to handle future budget deficits. OEMs must obtain a license --through Apple Vendor Support--for resale or distribution. Release date is expected to be in early fall with a suggested retail price of $150. The RPS programming manual is available separately for $30.

Printing Mailing Lists on One Line

by Don Norris

While Ken Silverman and I were returning from the Minneapolis Computer Show and Applefest, he asked me if it was possible to use Apple Writer /// Word Processing Language to print out Mail List Manager lists with all of the data on one line. Individual entries printed out from Mail List Manager will look like this:

```
Mr. Ken Silverman
International Apple Core
910A George Street
Santa Clara, CA  95050
```

Using the Apple Writer /// Utilities diskette and Apple Writer /// Word Processing Language I knew it was possible to do just what he wanted.

Thriving on a challenge such as this, I told him HELL yes. So here it is.

First, in order to use your Mail List Manager lists with Apple Writer /// you have to convert Mail List Manager files, which are stored as Pascal Data, to a text file which Apple Writer /// will be able to read. This is accomplished with the Apple Writer /// Utilities diskette. Boot the Apple Writer /// Utilities as described on page 103 of the Apple Writer /// manual. In response to the prompt, enter 3. Leave the Apple Writer /// Utilities diskette in Drive 1, press RETURN. The light under Drive 1 will come on for a few seconds, and then you will be prompted as follows:

```
MAIL LIST MANAGER TRANSFER
Drive number of Mail List Manager
diskette:

Enter 2

The next prompt you have is:

Name of Apple Writer /// Volume:

Enter .D1<RETURN>
```

Put the Mail List Manager data diskette into Drive 2, and the diskette you want the list transferred to into Drive 1. Press RETURN and the transfer of your Mail List Manager List will be made onto the diskette in Drive 1.

Boot Apple Writer ///. Use [O] (CONTROL O) to catalog the diskette you transferred your Mail List Manager List to and you will see that you now have a file on the diskette labeled MLMDATA. All Mail List Manager Lists that are transferred to Apple Writer /// text file format are labeled this way. So to avoid confusion you should change the name of the file using the [O] (CONTROL O) rename file option. For example, you could change MLMDATA to LIST1.

Assuming you are using the Mail List Manager Standard Format entries transferred from Mail List Manager, when loaded into Apple Writer ///, your Mail List Manager Lists will appear in the following format:

```
<1>
@1@Mr. Ken Silverman
@2@International Apple Core
@3@910A George Street
@4@Santa Clara,
^@5@CA
^@6@95050
[1]<2>
```

In order to put this onto one line you need to enter several automated text search and replace commands using [F] or, to do it even faster, you can write a WPL (Apple Writer /// Word Processing Language) program. [F] (Control F) is explained on page 36 in the Apple Writer /// manual.

Apple Writer /// Word Processing Language allows you to create a program which you store as a file, that will automatically make all of the changes necessary to printout your list on one line per entry. That is, you do not have to enter each command separately for each change, nor do you have to make notes for future use for changing other mailing lists to the same one line format.

For lack of a better name we will call this program Ken's one liner. In keeping with the SOS file name requirements it will be saved as: KENS.ONE.LINER.

To explain exactly how this program operates explanatory comments have been inserted. This is done using P followed by a space for inserting comments. Apple Writer /// Word Processing Language

ignores this, as explained on page 92 of the Apple Writer /// Manual.

In these explanatory comments the number in parenthesis refers to the page number in the Apple Writer /// manual which will provide you with additional information for your reference.

This program assumes that the list you want to print on one line has been loaded into memory.

P Every Mail List manager entry has the number of the entry at the beginning and ending of each entry. Since we do not want to print these out, they must be deleted.

BEGIN B

P This places the cursor at the beginning of memory

F*<+>,**A

P This line finds all of the first entry numbers on your list, such as <1> and deletes them as well as the RETURN, which follows the >. (42)

P Our next project is to remove the number at the end of each entry. This is done with the following line.

F<[=]<<A

P Now you need to delete @1@ and print the next lines of your mailing list on the same line. You can easily delete the @1@ with the following:

F<@1@<<A

P This will solve part of your problem. You still need to have all of your mailing list lines, 6 in this example, print out on one line.

P In order to print each succeeding line of your mailing list entry on the same line as the first you need to change the carriage return value on the print program values to Ø. This is done by embedding the following command into the text, ".CRØ".

P By changing CR1 to CRØ, in the Print/Program format values, your printer will not advance the paper one line when a carriage return is received. (58)

P Your margins for printer output will also be set with the text embedded command ".lm15" and ".rm95". Remember each text embedded command MUST be preceded and followed by a RETURN.

P Both of these changes can be accomplished with a single [F] command. Which will become the next line of KENS.ONE.LINER.

F<@1@<.CRØ>.LM15>.RM95><A

P This line says FIND @1@ and replace it with .CRØ and a RETURN everywhere it exists in memory from the beginning to the end. Remember the cursor is at the beginning of memory. (40)

P Additionally, to have your list printed in neatly formatted rows, you have to move your left margin. In this case I am moving the left margin over 25 spaces with the ".LM+25" text embedded command.

P Using [F] again I replace @2@, with .LM+25 and a RETURN character.

F<@2@<.LM+25><A

P Now the left margin must be moved over 30 more spaces to allow room for the company name. Additionally the"@3@" must be removed. The following line will accomplish this.

F<@3@<.LM+30><A

P Next the margin must be moved for the City name, as well as deleting the @4@. Here again [F] in the WPL program does this for us.

F<@4@<.LM+16><A

P The next two lines will move the margins and delete the unwanted characters for the State and ZIP code.

F<^@5@<.LM+3><A

F<^@6@<.CR1><A

P Notice that in this last line we have reset the Carriage Return entry in the Print/Program Command level to "1". This is because we now want the printer to execute a line feed with the carriage return.

P If you do not change CRØ to CR1 every one of the mail list entries will print out on top of the first one.

P Now that everything has been changed to the appropriate format lets print it out. This was the whole purpose of this program anyway. We do just that with the following line.

PNP

Continued on Page 22

VERSAFORM

Business Form Processor from Applied Software Technology

a review by Gene Wilson

Reprinted from the Cider Press

INTRODUCTION

This month we'll take a 'hard' look at a series of programs called "VersaForm". This 'program' is so extensive that it resembles software found on larger 'mini' computers, such as IBM's System 34/CMAS package (but we won't get into that here). VersaForm runs a hard disk version that takes 800+K. The review will be more like a two-pronged probe. I'll spend most of my time looking at what's available (in the form of a 'working' data base example). Woody Liswood has also provided some commentary in the form of an independent 'written review'.

Next month I'll look into the 'hard-disk' capabilities in more detail (assuming that my Corvus controller has returned from the 'factory'—one of these months I'll have to say a few words about 'controller' cards that have only one chip on a 'socket' on the entire board--but that will wait)!

CONFIGURATION

The most important part of 'getting' into VersaForm is to read and thoroughly understand the 'System Configuration Program'. The true power of VersaForm will not be seen unless things are done correctly here. There are provisions for assigning 'key-activated commands' while in data entry mode, and terminal initialization sequences. The user can even have a 'standard' printer initialization sequence, but still have the option of entering a 'different' sequence at print time. This allows printing some reports in 'compressed' mode, while others can be in any other 'font' or 'size/style' available on the printer (or interface card). The Configuration 'File' is named BJTERM. BJTERM must be on every disk which will be used as a 'boot' disk (DESIGN, FILING, REPORT, and CPRINT).

SYSTEM CONFIGURATION
*BJTERM

Prefix code for this terminal is: NONE

COMMAND	CODE	PREFIXED
Display the cmd menu	63	No
Validate	3	No
Get a form	7	No
Save a form		No
First form in file		No
Last form in file		No
File space report	18	No
Print current form	16	No
Calculator	3	No
Next form	93	No
Back to previous form	2	No
Clear to blank form	17	No
Erase unvalidated data	5	No
Page forward	62	No
Page backward	60	No
Remove the current form	15	No
Index list	9	No
Delete a line	4	No
Cursor to command line	27	No
Quit - exit the program	25	No

Video reverse type : 0 (none)
Video reverse seq : NONE
Normal video seq : NONE
Terminal initialization seq: NONE

Printer page width default : 132
Printer page length default : 66

As important as the 'Configuration' process is, it's the LAST chapter in the User's Guide. (The Guide is a document that you should read one time then put it carefully away—far, far away.) There is a very good Tutorial Disk, a 'Hands-On Experience' Manual, and a very detailed 'Reference Summary'. Don't look for configuration information in the Manual or the Summary—it just isn't there!

PHILOSOPHY

This isn't a major problem. The whole series of programs called VersaForm are ALL set up with the 'non-computer-type' in mind. The menus are clear and usually concise. The decision process is straightforward, and there is ample opportunity (in most cases) to 'back out' of a bad decision. There need be no fear of making a mistake that can't be rectified without losing many hours of work. This quick experimentation can be a boon to the person who isn't entirely sure about the final 'look' or 'appearance' of a report.

```
Printer does not have form feed
Printer does take LF after CR
Printer initialization seq:   27  69
Operator will be asked for
printer control sequence.

Display dummy data character is "."
Printer dummy data character is "-"

Program volume names :
Vol name for Design pgm is DESIGN  .
Vol name for Filing pgm is FILING  .
Vol name for Report pgm is REPORT  .
Vol name for Copy/Prnt pgm is CPRINT  .
Vol name for Rptwrk disk is RPTWORK .

Default vol name for files is #5      .

Diagnostic mode is not set.
```

There are many other Data Base Programs on the market. VersaForm fills a particular function that lies somewhere between the very rigidly structured formats of DB Master, and the mind bending task of learning another 'language' with dBase II. The other programs are great, but only VersaForm has allowed me the freedom to take a 'quick shot' at form design or entry screen formats. If I don't like the end result, it only takes a couple of minutes to 'make it better'.

```
File #11:TRUCK

File size: 1040 storage units.
Minimum form size: 110 characters, 1 storage units.
Column line size: 0 characters
Estimated file capacity: 936 forms with no column lines.
```

DESIGNING THE FORM

This is the easiest part of the whole system. Simply decide what goes where, answer questions regarding range checking, length, justification, if item is mandatory, look-up table, calculations, and automatic filling, etc. The program will ask all the right questions, keep track of the answers, and even give the user hard copy of the whole session. Changes are easy to make, and the input 'mask' can be made quickly (changed, modified, or drastically altered just as quickly!).

```
& -- Any letter of the alphabet is OK
# -- Any digit (0-9) is OK
/ -- Optional leading digit ( 0-9 )
? -- Any character is OK

Any other characters mean that only
the particular character will be
accepted in that position.


Format &&------------------


This item: Job-----------------
```

```
Truck Ticket Entry Form:
                                                    Semi          10Whlr
Job -- Ticket # ------ Mo -- Day -- Truck # ------ Hrs1 ------ Hrs2 ------
Code -                                              Rate          Rate
       Owner ---------------- Driver -------------- Tr#1  49.50 Tr#2  46.30
Gross Amt ----------                                Amt1 ------ Amt2 ------
Less Brok ----------
Less PUC. ----------
          ==========
Net Due.$ ----------
```

CHECKING AND AUTOMATIC FILLING

```
Minimum-length 1-    Maximum-length- 2-
Justify-(L/R/#) r    Selfchecking (Y) -
Numeric --- (Y) y    Date ------- (Y) -
Yes-or-no --(Y) -    Mandatory -- (Y) y
```

EXTENDED CHECKS

```
Ranges (Y) y         List (Y) -
Format (Y) -
```

AUTOMATIC FILLING

```
Lookup (Y) -           Todaysdate (Y) -
Calc - (Y) -       Column Total (Y) -

This item: Mo------------------
```

RANGE CHECKING

On each line type the limits of one range. Input will be OK if it falls in any one range. If you give only the low (high) value, any input higher (lower) will be accepted.

```
          Low value          High value
L#  L.                  H.
01  1-                  12

--
This item: Mo------------------
```

TABLE LOOKUP

The data for this item will be obtained by looking it up in the table below. The item to get the lookup value from is: Item name Truck-#----------

```
     Look up          Result
L# ....L.....  .......R.......
01 J-1-------  BIG-'J'-TRKG---
02 J-2-------  BIG-'J'-TRKG---
03 J-3-------  BIG-'J'-TRKG---
04 J-4-------  BIG-'J'-TRKG---
05 KB-1------  BISHOP---------
06 B-21------  BLANCHARD------
07 AC-1------  AL-CAIN--------
08 H-88------  CIRCLE-'H'-----
09 T-1-------  DAVID-ROGERS---
10 DDL-1-----  DDL-TRKG-------
11 DDL-6-----  DDL-TRKG-------
12 DDL-7-----  DDL-TRKG-------
13 H-10------  HIGDON---------
14 H-11------  HOSKINS--------
15 K-99------  P&K-TRKG-------
16 K-111-----  P&K-TRKG-------
17 R-15------  ROGERS---------
18 R-18------  ROGERS---------
19 R-83------  ROGERS---------
20 O-1-------  ROGERS---------
21 R-97------  ROGERS---------
22 505-------  TEDS-TRKG------
23 T-1-------  THOMAS---------
24 G-1-------  GORDON---------
25 G-2-------  GORDON---------

This item: Owner---------------
```

We see that the item MOnth has a minimum length of one numeric character, a max. limit of two, the number is right justified and it is mandatory (you can't leave the item blank and then save the record). Low acceptable value is 1 and high value is 12. Any number (or character entry) out of this range will not be accepted.

Table lookup shows that automatic filling of Owner and Driver are both tied to the truck number. In this data base, every truck number is unique. This saves a great deal of time if much of the data can be keyed to some common (but unique) item. If there are over 99 items, the lookup table is not going to provide you with a valuable service (a limitation!). The operator can enter a value different than the one provided by the lookup table.

The Net-Due item is calculated by the Gross-Amt less the Less-Brok item, less the Less-Puc item. Round off can be accomplished here as well. Dummy (intermediate) items can be set up to accomplish further manipulation of the data.

CALCULATION

The calculation may be made by adding, subtracting, multiplying, or dividing two items, or one item and a number.

Operations are +,-,*,/,Low(L),High(H)

```
Operations     Items/numbers
              I1: GROSS-AMT-----------
   OP1: -     I2: LESS-BROK-----------
   OP2: -     I3: LESS-PUC.-----------
   OP3: -     I4: --------------------
   OP4: -     I5: --------------------
   OP5: -     I6: --------------------

This item: Net-Due.$-----------
```

```
     Look up          Result
L# ....L.....  ......R.......
01 J-1-------  BELL----------
02 J-2-------  CAMPBELL------
03 J-3-------  BREAD---------
04 J-4-------  LUCAS---------
05 KB-1------  KARL-BISHOP---
06 B-21------  L.-ANDERSON---
07 AC-1------  AL-CAIN-------
08 H-88------  J.-CHRIS------
09 T-1-------  D.-ROGERS-----
10 DDL-1-----  BULLOCK-------
11 DDL-6-----  W.-SMITH------
12 DDL-7-----  BULLOCK-------
13 H-10------  J.-HIGDON-----
14 H-11------  BEN-HOSKINS---
15 K-99------  K.E.-JOHNSON--
16 K-111-----  KINNARD-------
17 R-15------  R.T.-EDWARDS--
18 R-18------  B.-WELSCH-----
19 R-83------  SHELTON-------
20 O-1-------  O.-WEBB-------
21 R-97------  APARICIO------
22 505-------  T.-FORD-------
23 T-1-------  ROGER-THOMAS--
24 G-1-------  D.-MARSHALL---
25 G-2-------  V.R.-GORDON---

This item: Driver--------------
```

FILING

No secrets here. Hard work is the only answer. Just keep dumping data into the machine. The program will make things as tolerable as possible, and a lot of checking can be done to see that the data entry is done properly. Hard copy of the day's activity can be obtained as well. There is no reason to have faulty data if a reasonable system of checks is maintained. The data entry does not have to wait long periods while the screen reformats between records. The program is fast in this area and won't slow you down.

```
Truck Ticket Entry Form:
                                                        Semi          10Whlr
Job SA Ticket # 1359-- Mo -5 Day 16 Truck # 505---  Hrs1 --8.25  Hrs2 ------
Code B                                              Rate          Rate
       Owner TEDS-TRKG------ Driver T.-FORD-------  Tr#1   49.50  Tr#2   46.30
Gross Amt ---408.38                                 Amt1  408.38  Amt2 --0.00
Less Brok ----20.42
Less PUC. -----0.92
          =========
Net Due.$ ---387.04
```

1241	OWNER:	CIRCLE 'H'	SEMI:	6.00 @ 49.50 =	297.00	GROSS AMT:	297.00
JOB:SA	DRIVER:	J. CHRIS	10WL:	@ 49.50 =		LESS BROK:	14.85
5/10/	TRUCK#:	H-88				LESS PUC:	0.67
CODE:C						NET DUE: $	281.48
1359	OWNER:	TEDS TRKG	SEMI:	8.25 @ 49.50 =	408.38	GROSS AMT:	408.38
JOB:SA	DRIVER:	T. FORD	10WL:	@ 49.50 =	0.00	LESS BROK:	20.42
5/16/	TRUCK#:	505				LESS PUC:	0.92
CODE:B						NET DUE: $	387.04
1361	OWNER:	TEDS TRKG	SEMI:	3.00 @ 49.50 =	148.50	GROSS AMT:	148.50
JOB:SA	DRIVER:	T. FORD	10WL:	@ 49.50 =	0.00	LESS BROK:	7.43
5/17/	TRUCK#:	505				LESS PUC:	0.33
CODE:B						NET DUE: $	140.74
1945	OWNER:	HIGDON	SEMI:	7.75 @ 49.50 =	383.63	GROSS AMT:	383.63
JOB:SA	DRIVER:	J. HIGDON	10WL:	@ 49.50 =	0.00	LESS BROK:	19.18
5/30/	TRUCK#:	H-10				LESS PUC:	0.86
CODE:C						NET DUE: $	363.59
2175	OWNER:	ROGERS	SEMI:	4.50 @ 49.50 =	222.75	GROSS AMT:	222.75
JOB:SA	DRIVER:	SHELTON	10WL:	@ 49.50 =	0.00	LESS BROK:	11.14
5/11/	TRUCK#:	R-83				LESS PUC:	0.50
CODE:B						NET DUE: $	211.11
2187	OWNER:	GORDON	SEMI:	7.75 @ 49.50 =	383.63	GROSS AMT:	383.63
JOB:SA	DRIVER:	D. MARSHALL	10WL:	@ 49.50 =	0.00	LESS BROK:	19.18
5/15/	TRUCK#:	G-1				LESS PUC:	0.86
CODE:B						NET DUE: $	363.59
2188	OWNER:	GORDON	SEMI:	7.25 @ 49.50 =	358.88	GROSS AMT:	358.88
JOB:SA	DRIVER:	V.R. GORDON	10WL:	@ 49.50 =	0.00	LESS BROK:	17.94
5/30/	TRUCK#:	G-2				LESS PUC:	0.81
CODE:B						NET DUE: $	340.13

Da	Owner	Truck	Driver	Ticket	Hrs1	Hrs2	Gross Amt	Less Brok	Less PUC.	Net Due.$
	ROGERS				61.25	0	3031.90	151.60	6.82	2873.47
2	TEDS TRKG	505	T. FORD	2407	5.00		247.50	12.38	0.56	234.57
10	TEDS TRKG	505	T. FORD	2406	8.00		396.00	19.80	0.89	375.31
11	TEDS TRKG	505	T. FORD	2405	5.00		247.50	12.38	0.56	234.57
15	TEDS TRKG	505	T. FORD	2403	8.00		396.00	19.80	0.89	375.31
16	TEDS TRKG	505	T. FORD	1359	8.25		408.38	20.42	0.92	387.04
17	TEDS TRKG	505	T. FORD	1361	3.00		148.50	7.43	0.33	140.74
	TEDS TRKG				37.25	0	1843.88	92.21	4.15	1747.54
11	THOMAS TRKG	T-01	ROGER THOMAS	2559		4.75	219.93	11.00	0.49	208.44
	THOMAS TRKG				0	4.75	219.93	11.00	0.49	208.44
	REPORT CONTROL INSTRUCTIONS				351.25	10.25	17861.56	893.08	40.20	16928.28

FILE #11:TRUCK, REPORT TEMPT1

```
Page width 132
Page length 66
Title:
Print:
     Day
     Ticket #
     Owner
     Driver
     Truck #
     Hrs1
     Hrs2
     Gross Amt
Sort:
     Day                    ( ascending)
     Owner                  ( ascending)
     Truck #                ( ascending)
Subtotal control:
     Day
Total:
     Hrs1
     Hrs2
     Gross Amt
```

8/15/82

Da	Ticket	Owner	Driver	Truck	Hrs1	Hrs2	Gross Amt
			K.E. JOHNSON	K-99	6.25		309.38
					6.25	0	309.38
10	2554	BIG 'J' TRKG	CAMPBELL	J-2	7.50		371.25
10	2557	BLANCHARD	L. ANDERSON	B-21	1.00		49.50
10	1241	CIRCLE 'H'	J. CHRIS	H-88	6.00		297.00
10	3493	DDL TRKG	W. SMITH	DDL-6	7.75		383.63
10	2555	GORDON	V.R. GORDON	G-2	7.50		371.25
10	12174	ROGERS	SHELTON	R-83	8.25		408.38
10	2406	TEDS TRKG	T. FORD	505	8.00		396.00
10					46.00	0	2277.01
11	2504	AL CAIN	AL CAIN	AC-1	2.00		99.00
11	2558	BIG 'J' TRKG	CAMPBELL	J-2	4.75		235.13
11	2422	GORDON	D. MARSHALL	G-1	2.00		99.00
11	2175	ROGERS	SHELTON	R-83	4.50		222.75
11	2405	TEDS TRKG	T. FORD	505	5.00		247.50
11	2559	THOMAS TRKG	ROGER THOMAS	T-01		4.75	219.93
11					18.25	4.75	1123.31

```
01       1         2         3         4         5         6         7         8         9
02 456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
03
04       1         2         3         4         5         6         7         8         9
05 456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
06
```

```
******     OWNER:  *************** SEMI:  ****** @ 49.50 =  ******     GROSS AMT:*********
JOB:**     DRIVER:   **************1OWL:  ****** @ 49.50 =  ******     LESS BROK:*********
**/**/     TRUCK#: ******                                               LESS PUC: *********
                                                                        ==================
CODE:*                                                                  NET DUE: $*********
```

REPORTS

The standard reports are handled well. Simply move the cursor around the entry screen and indicate which items will be printed (and the order of printing). Subtotals, item totals, and sorting criteria are all established in a fast, straightforward manner. Changes are easily made.

The real challenge is to design a 'pre-printed' form, one which has very critical locations for data. The tools provided include a numeric scale and a 'dummy' record can be printed to the new form with data represented as '*'s. Maximum fields are laid out so no doubt exists as to what will appear. Text can be added to every printed form to further clarify the data that is output.

The whole idea of having a data base is to be able to store and manipulate the data. The report module allows the data to be displayed in almost any imaginable format, on pre-printed forms, in many different sorting orders, and different combinations of data can be readily compared.

SUMMARY

VersaForm is a valuable tool. While not capable of handling the 'huge' data base applications occasionally required, it DOES handle the bulk of jobs that exist. The program is fast, efficient, well written, and well executed. The User's Guide randomly seeks to provide information, and should only be used if all other means fail. The program will prove to be a real joy from the standpoint of just 'powering-up' and 'creating'. This is one of the few programs that deserves to be on every business user's shelf.

Reprinted from the Cider Press

Versa Form

Business Forms Processor

Published by:

Applied Software Technology
15985 Greenwood Road
Monte Sereno, Ca. 95030

Systems:
Apple II and ///
Also Hard-Disk versions

VersaForm does just that. It creates, and lets you use almost any type of form that you might need. Invoices, packing slips, point-of-sale receipts, personnel records are all easy to create and use. The flexibility shown by the program also allows you to create "mini data management systems" (my quotes). However, before you jump into the data management area you should create and use a few form applications so that you are familiar with the way the program thinks.

VersaForm comes packaged in a box with a somewhat easy-to-read and use manual, very useful reference summary, a hands on experience book (which you should definitely do first) and six disks. Yep, 6 separate disks.

FEATURES

VersaForm is a PASCAL based system. It stores it's data as "forms" rather than what a data management system would call a record. These forms can have single items, such as name, date, address, etc. They also can have column data such as part number, description, etc. There is also the capability to have a data entry mask to check data as it is entered. You can require fields that must be filed in, have only numbers, dates, special formats such as phone numbers, specifically formatted numbers, social security numbers or special types of part numbers. You can also have a list of separate entries which can be allowed or you can specify a range for the entry value.

There is automatic calculation during data entry and printing. You can, for example, enter in a part number, VersaForm will then look up and enter in the description and the price, then multiply the price by the number purchased and put that in the cost column. Then, when that invoice is completed, it will automatically compute the tax and total the entire purchase and print the invoice for your customer.

Since this is PASCAL, VersaForm also works with your 80 column boards so that you can have what you see is what you get type of form to use.

According to the documentation (I did not test) it creates standard Apple Pascal files so that you can use your data in other programs.

EASE OF USE

I found the program rather easy to use. It, however, has so many features that you should spend a session or two with the manual, the hands-on-experience disk and try a thing or two, before you jump in and start a important application.

The only part of the program, in my opinion, which will cause some head-shaking is the report generator. That is not because it doesn't work, but because it gives you many, many options. And the Documentation does not always seem to be clear.

CRITICISMS

I could not find any problems with the program. No bombs, crashes or any of those things. When you use it to do forms, you will be very happy. When you use it as a data management system, you will need to be careful in your conceptualization otherwise you will find that it is not doing things just quite the way you expected.

My only criticism is that it does not do sub-total page breaks in the report writer. This made the program less that useful for creating reports when I was trying to use this as a data management system rather than a form generator. Otherwise, like I said, no problems.

DOCUMENTATION

The documentation is well written. It is, however, rather hard to find things in. Many important points are just kind of mentioned here and there. Things like maximum number of records, can the files span multiple disks, and other important goodies are not all in one place. The best way to approach this manual, is to read it once. Then sit down and go the examples in the manual while you are sitting at your computer. Make certain that you have a yellow high-liter and a pencil. You will need to make notes as to the features. And, since the instructions are scattered all over the place, you should hi-lite the commands with the yellow pen. Then, when you need to refer back to the manual you can find what you need. I didn't the first couple of times I used the program. And I wasted much time trying to find things. The problem is that this program has so many features, you will need to spend lots of time with it before you have memorized all of the commands.

HOW-IT-WORKS

The six disks which come with the system are:

1. Design program disk.
2. Filing program disk.
3. Report program disk.
4. Copy/Print program disk.
5. Tutorial disk.
6. Report work disk.

CREATING A FORM

The forms generator is a pleasure to use. You put the Forms design disk into the drive and a Pascal initialized disk into drive two and away you go. Only now you are using drives #4 and #5 because Pascal uses those external device names for the first two disk drives. If you need to, you will have to initialize some disks for use. There are specific instructions about how to do that so don't worry.

In fact, when you boot the forms design disk, you will be given a 4 item menu. Forms design, System configuration, Initialize diskettes or name diskette.

When you chose Forms Design, you go to another four item menu for copying an existing design, changing a existing form, designing a new form, or printing a form definition.

VersaForm will let you place Text or what they call Items anywhere you want on the screen. Text is the labels you have for your data fields. Items is the actual data field. You do your forms design by moving the cursor around the screen and typing letters and dots. The dots indicate a data field. Here's an example.

If you wanted a heading at the top of your page, you would cursor over and type in the heading. It could be your letter-head, or whatever.

Then, you might say:

Bug Report

There is a bug which occurs in Mail List Manager when one tries to filter a mail list. Should you--for example--want to filter those names from a list of 300 of computer owners that own an Apple ///, Mail List Manager will for some unexplained reason will truncate the output so that you end up with an incomplete list (typically the last few names are deleted).

As soon as Apple finds the cure for this bug, it will notify all Mail List Manager owners.

One reader, Craig Stauffer from Sunnyvale, contributed this error in the Universal Parallel Interface Card manual: on page 9 in the third paragraph from the bottom the sentence reads....and the Add a Driver function.... This should read "Read a Driver". Add a Driver is old SOS terminology used in early versions. The Silentype /// manual--which is the oldest manual in the Apple /// system--for example refers to "Add a Driver" countless times.

VisiSchedule
If you use a period in a volume (diskette) name, VisiSchedule will not be able to load any files you have saved on the diskette. This means you have to boot the Utilities diskette and rename the volume in order to use the files you have saved.

SOS Pathname conventions allow the use of a period in volume names. The VisiSchedule manual on page 2-16, has an example of a volume name with a period in it. Hopefully VisiCorp will correct this and follow the SOS Pathname conventions, rather than establishing their own conventions. By following the SOS pathname conventions it will help make all Apple /// software more user friendly.

Printer set up strings for setting the left margin do not work. To set up the left margin using a Qume Sprint 9, boot VisiCalc. Then with the printer on, enter the following: /PP"(then the number of spaces you want for the left margin)<RETURN>. The printer head will now move over to your desired left margin. As long as you leave the printer ON and do not RESET it, the printer will always use this new setting for a left margin.

Quick File ///

This new file management program is said to provide users with a filing system to help in managing small or medium-sized databases of less than 600 records on the Apple ///. Aimed at users such as doctors, owners of small businesses, or scientists, the program allows simple arrangement of records in alphabetic, numeric, or chronological order. For personal use Quick File can keep track of investments, collections, personal inventory and financial records. The program is divided into three menus: Main, File and Report. The user interface has been improved by allowing one to view two records by simply pressing two keys to switch between displays. Furthermore the program makes extensive use of the Open Apple key plus a letter to execute functions (Open Apple + A arranges records in file). In addition to the above Quick File has the following features:

- --simple form design
- --users can add or delete categories without having to re-input stored data (15 cat. max)
- --Quick File can search and display records by chosen categories
- --records can be displayed or summarized simultaneously (one derived column)

The cost of Quick File /// is $100.00

The Third Basic

By: Taylor Pohlman
Reprinted from Softalk Magazine

When last we left the Lone Ranger, huge boulders were crashing down the slope toward his tiny campfire....No, I'm sorry to say that the September column wasn't quite that breathtaking or cliff-hanging. However, taking the Apple /// out for a spin does excite a lot of people, and we hope that includes you. If you haven't read last month's column, we recommend you get a copy. This series is progressive in that each article builds on the previous one. We're going to assume that you have been following the series, so that each month we can cover a new topic in the least possible amount of purple prose.

Another reminder before we start: we welcome questions and comments on this series or on Basic in general. Because of deadlines, each article is being written before the previous month's is in print. Thus, reaction to your timely comment will be somewhat delayed. Let those cards and letters roll in, and the responses will show up just as soon as inhumanly possible.

The SOS File System Revisited. After a brief discussion of the Apple /// SOS file system, last month's column concluded with something of a challenge for you. We were working with a program to dump the contents of the screen to the Silentype printer, and we mentioned that the program could be generalized for any file, including text files. The point was that SOS takes care of all the details about how each device works, so the user can change things at will. For reference, here's the program with which we were working:

```
50      OPEN#1,".silentype"
90      INVOKE"readcrt.inv"
150     FOR vp=1 TO 23
155       VPOS=vp
160       FOR hp=1 TO 80
165         HPOS=hp
170         PERFORM readc@value%)
180         PRINT#1;CHR$(value%);
190         NEXT hp
200     PRINT#1
210       NEXT vp
900     VPOS=23:HPOS=1
1000    END
```

Before we modify this program to generalize it, did you try to simplify the program by using VPOS and HPOS directly in lines 150 and 160? By that we mean:

```
150 FOR VPOS=1 TO 23
160 FOR HPOS=1 TO 80
```

If you did, you know that Basic will respond to this change with the classically familiar syntax error, because VPOS and HPOS are reserved words and cannot be used as index variables.

To continue, the challenge was to generalize the screen dumping program so the output could go to any file. Here's one solution to that problem:

```
50      VPOS=23:HPOS=1
60      INPUT"Name of file to dump
        screen to:";filename$
100     OPEN#1,filename$
110     INVOKE"readcrt.inv"
120     FOR vertical=1 TO 23
130       VPOS=vertical
140       FOR horizontal=1 TO 80
150         HPOS=horizontal
160         PERFORM readc@value%)
170         PRINT#1;CHR$(value%);
180         NEXT horizontal
190       PRINT#1
200       NEXT vertical
210     CLOSE
300     VPOS=23:HPOS=1
310     END
```

Several differences are worthy of note. First, the cursor has been repositioned in line 50 to the bottom of the screen to avoid overwriting any existing data. The user is then prompted in line 60 to type in the name of the output file. Note that this can be any filename legal on the Apple /// that accepts output (printers, the communications port, a disk text file, even .CONSOLE itself).

Note also the addition of the close statement at line 210. This ensures that all files are properly written to and dispensed with at the conclusion of the program. Failure to close files properly can leave some data still in memory (since files aren't automatically closed at the end of the program). This can have some interesting consequences if the file in question is a disk file and you switch to

another diskette that doesn't have the file created on it. Now is the time to form the habit of closing all files at the end of a program.

Running this program can be instructive. Obviously, if you reply ".SILENTYPE" to the prompt, it will work like the first example. Try replying ".CONSOLE" now. After the usual initial whirring of the disk to load the Invokable Module, the program appears to go to sleep for forty seconds or so. What's happening is that the program is reading a character and then copying it back on top of itself! The Apple /// is working its little heart out, and the result is as exciting as watching bread mold.

Now try replying with a disk file name (you can just make up a name, as long as it follows the filename rules). The disk will whir as before. This time Basic has a number of jobs to do. First, it must open the disk file using the name you gave it (let's assume you typed MYFILE.SCREEN). Basic tells SOS to create the file (assuming it doesn't already exist), by making an entry in the directory of the current disk volume and finding initial space for the file. Basic then sets up a buffer area in memory for communication of data to and from the file. Since Apple /// divides the disk up into blocks of 512 characters, this internal buffer is 512 bytes. This buffer size is fixed no matter what record size you specify. Later on in this article, we'll look at techniques that use that piece of information to ensure maximum efficiency and performance in disk-based application programs.

Once the file is opened, Basic than invokes the Readcrt module, and execution begins. Notice that, although the printer in our precious example started almost immediately, there is a noticeable pause before the disk spins into action, and it appears to spin only four times before the program stops. What's happening is this: line 170 prints one character at a time into the buffer. After eighty characters, line 190 prints a carriage return into the buffer and then starts the next line. After a little more than six lines of the screen (480 bytes plus six returns plus 26 bytes of line 7 to be exact) the 512 byte buffer is full and must be written to disk. That's the first spin of the disk that writes the first block of the file. Next, the block number is incremented, and more writing starts from line 7 of the screen. 512 bytes later the same process is repeated until all the screen is read by the program and written into the last buffer. Some arithmetic would convince you that Basic is in the middle of its fourth buffer when the program finishes reading line 23 of the screen. That's when the previous comment about being sure to close files comes in handy. The Close command in line 210 forces the current buffer to be written to disk, even if it's not full, and the directory entry is updated to reflect the new file information.

After running the program, the catalog listing of the file should look something like figure 1.

```
TYPE    BLKS     NAME        MODIFIED TIME

TEXT  00005  MYFILE.SCREEN 00/00/00 00:00

-------------------------------------------

      CREATED TIME       EOF

      00/00/00 00:00    1863
```

Figure 1.

Notice that Basic identified the file as a text file automatically, because the Print# command was used to write to it. Notice also that the Blks used column shows five. That disagrees with what we had predicted, since the screen data should have been able to fit into four blocks (2048 bytes). The reason for the extra block is that SOS allocates an extra block as an index block to store information about where the rest of the blocks in the file are physically located. This ensures that a large file can be created, even if the disk is fragmented into small areas of unused space. If you look closely at the directories of various files, you will note that all of them have one more block that the EOF column would indicate, except for the one-block files, which have no need of an index block. In this case, the EOF (end of file) is after 1863 bytes. That works out to twenty-three lines of eighty characters (1840 bytes) plus twenty-three carriage returns for a total of 1863 bytes. "Close enough for folk music," as they used to say in high school.

One last subject before we move on to further explore files. The Silentype gave us a permanent record of what was on the screen, but since we wrote the results to a disk file this time, we need a way to dump the contents of MYFILE.SCREEN to the printer. The following program easily accomplishes the task and serves as a general file-to-file transfer program:

```
5     INPUT"Name of file to dump:
      ";inputfile$
10    OPEN#1,inputfile$
15    console=0
20    INPUT"File to dump to:
      ";outputfile$
25    OPEN#2,outputfile$
30    check$=MID$(outputfile$,1,3)
35    IF check$=".co"OR check$=".CO"
      OR check$=".Co"Then console =1
40    IF console THEN HOME
45    ON EOF#1GOTO 65
50    INPUT#1;a$
55    PRINT#2;a$;
60    GOTO 50
65    IF console THEN HPOS=1:VPOS=23
70    CLOSE
75    END
```

There. AS long as you don't try to read from the printer and print to the keyboard, it should work fine.

Note that we've checked in line 35 to see if the device being written to is .CONSOLE. If so, line 40 clears the screen to reproduce exactly what was there when the original program was run. Line 65 repositions the cursor to the bottom of the screen so that the prompt will not cause the top line to scroll out of view.

More on Files. The subtle and nefarious purpose of this lesson, if you haven't realized by now, is to provide more insight into Business Basic disk files. We've remained true to the promise of the first article and assumed that you are skilled in Basic, so hang on as things get more interesting....

So far, we've considered only the type of disk files referred to as text files. These are files that contain ASCII characters, which are representative of what would be printed out if we wrote data to the screen instead of disk. For now we'll stick with this file type and later touch on data files, a useful and relatively unique file type on the Apple ///.

We've already learned that the disk is organized into 512-byte blocks. In fact, Basic text file records can be of any reasonable size. Instead of using the Open statement, which assigns a default of 512 bytes, we could have used the Create statement, which allows up to 32,767-byte records to be used. Of course, the record size of a particular file is of no consequence if we are merely going to read each string in order (as we did with the contents of the screen).

The real power of creating files of various record sizes is to be able to read data on a particular item in the film randomly without having to deal with the other data in the file. For example, if we had wanted to print the twenty-first line of the screen in the previous example, it would be necessary to input the first twenty lines, discard the data, and then finally read and print the line we wanted. A much more efficient way would be to create the file as a random access file with record size of eighty-one bytes. Since each record will correspond with one line of the screen, we have an easy way too address the data in question. Compare the examples below with the previous sequential access examples:

```
50    VPOS=23:HPOS=1
60    INPUT"Name of file to dump
      screen to: ";filename$
70    CREATE filename$, TEXT,81
100   OPEN#1,filename$
110   INVOKE"readcrt.inv"
115   cum$=""
120   FOR vertical=1 TO 23
130     VPOS=vertical
140       FOR horizontal=1 TO 80
150       HPOS=horizontal
160       PERFORM readc@value%)
170       cum$=cum$+CHR$(value%)
180       NEXT horizontal
190     PRINT#1,vertical;cum$
195     cum$=""
200     NEXT vertical
210   CLOSE
300   VPOS=23:HPOS=1
310   END
```

Note that we have added line 70 to create the filename with the proper record size. The notation of Text is extra baggage, since the Print statements in the program will automatically define it as a text file, but it is good practice to be specific. I have also added a new wrinkle

in lines 115, 170, 190, and 195. Instead of printing each character as it is read, the variable "cum$" is used to accumulate characters as they are read from the screen. Line 190 prints the entire line of the screen using the vertical position as the record number. The result when running this program seems the same as when running the sequential version, except for one thing. If you catalog the resulting filename, it should look something like figure 2 (assuming a name of SCR.DUMPL.RND).

```
TYPE    BLKS     NAME          MODIFIED TIME

TEXT  00005  SCR.DUMP.RND 00/00/00/ 00:00

-------------------------------------------

      CREATED TIME      EOF

      00/00/00 00:00    1944
```

Figure 2.

Everything is the same except the length. It turns out that, when a file is created, the first record is record 0, not record 1. This is consistent with the first element of an array being element 0. Therefore Basic has reserved twenty-four (not twenty-three) records of eighty-one bytes each for a total of 1944 bytes.

Now that we have associated a record number with every line on the original screen, we can locate a given line by just giving its number instead of having to read through all the other lines to find it. Witness the modified read program:

```
5       INPUT"Name of file to dump:
        ";inputfile$
10      OPEN#1,inputfile$
15      console=0
20      INPUT"File to dump to: ";
        outputfile$
25      OPEN#2,outputfile$
30      check$=MID$(outputfile$,1,3)
35      IF check$=".co"OR check$=".CO"
        OR check$=".Co" THEN console=1
40      IF console THEN HOME
45      ON EOF#1 GOTO 65
47      INPUT"record number to dump:
        ";rec
48      IF rec=0 THEN 65
50      INPUT#1,rec;a$
55      PRINT#2;a$;
60      GOTO 47
65      IF console THEN HPOS=1:VPOS=23
70      CLOSE
75      END
```

This program is very similar to the previous program except that line 47 asks for the specific record to dump, line 48 gives us a way out by checking for zero, and line 50 has been modified to read directly to the record number previously entered.

Some experimentation with this program will produce interesting results. Try reading records 1,6,12, and 18. In each case, you will cause a disk access (whirring is a clue) to read the particular record. Now try reading records 6,7,8,9, and 10 in any order you choose, The first record you read will probably cause a disk access, but the others should occur virtually instantaneously without causing disk activity. This is because SOS is still buffering flies in 512-byte blocks, and all those records fall within one block. There was no need to reread the disk because the data was already in memory. Careful planning of your record sizes and reading sequences can have the effect of substantially increasing the performance of your program, if as many reads as possible occur within the current buffer.

One interesting postscript before we proceed: If you ask for record 6 there will typically be a disk read, as we've said. If you immediately request record 5, another disk read will be performed. This is what you might expect, but more is going on here than meets the eye. Simple calculation will prove that record 6 actually occupies space in both block 1 and block 2 of the file. The first six records, 0 through 5, occupy 6*81 or 486 bytes of the first block, leaving only twenty-six bytes in that first block for record number 6. The remaining fifty-five bytes are in block 2.

Thus a read to record 6 actually triggers two disk reads, one to load in block 1 for the first part of record 6, and one for block 2 to obtain the remainder of the record. Therefore, when you requested record 5, Basic had to go back and reread block 1 (remember, only one block is kept in memory per file).

A little more arithmetic will show which other records are in this same situation. The moral is simple: if possible, make your record sizes such that they evenly divide into 512 or are a multiple of 512. That may waste a little space, but the waste may be more than compensated for in the ability to predict when disk access will take place.

A Final Challenge. We just reviewed the last five or six paragraphs and discovered that our usual humorous style has been replaced by long, detailed discourses of unrelieved tedium. There is, unfortunately, no letup in sight.

To this point we have been using "record number" files (called random access by most people) with record numbers that span a rather narrow range. SOS permits random files to have record numbers in the range of 0 to 32767. However, SOS does not demand that a file actually have all the records present on the disk. Records are allocated as written, with only a little space taken up to keep track of where everything is. To illustrate the power this gives, consider the following problem:

A distribution company wants to keep track of their part numbers and descriptions. The part numbers are four-digit numbers. Following is a simple program to create the part number file.

Between now and next time, you could try writing a program to retrieve part number information randomly and make changes as required. Without further ado...

```
5       HOME
10      PRINT"Parts file Create and
        Add program"
20      PRINT
30      PRINT"Type 1 to Create a parts
        file":PRINT
40      PRINT"Type 2 to add to an
        existing parts file"
50      PRINT:INPUT"Your selection:";
        a$
60      IF a$=""THEN 1000
70      a=VAL(a$)
80      ON a GOTO 100,400
90      GOTO 5
100     PRINT:INPUT"name of new parts
        file:";a$
110     IF a$="" THEN 5
120     CREATE a$, TEXT,64
130     PRINT"Parts file ";a$;
        "created."
140     GOTO 5
400     PRINT:INPUT "Name of existing
         parts file: ";a$
410     IF a$="" THEN 5
420     OPEN#1,a$
430     HOME
500     PRINT:INPUT"Part number to
        add:";a$
510     IF a$="" THEN 5
520     a=VAL(a$)
530     IF a<1 OR a>32767 OR INT(a)<>a
        THEN 500
535     rec=a
540     rec$=a$+"®"
545     PRINT:INPUT"Description: ";a$
550     IF LEN(a$)>30 THEN a$=MID$
        (a$,1,30)
560     rec$=rec$+a$+"®"
570     PRINT:INPUT"location:";a$
580     IF LEN(a$)>10 Then a$=MID$
        (a$,1,10)
590     rec$=rec$+a$+"®"
600     PRINT:INPUT"Quantity on hand:
        ";a$
610     a=0: a=VAL(a$): IF INT (a)<>a
        THEN 600
620     rec$=rec$+a$+"®"
630     PRINT:PRINT"Record is:";rec$;
        " OK? ";
640     INPUT"";a$
650     a$=MID$(a$,1,1):IF a$<>"y"
        AND a$<>"Y" THEN 430
660     PRINT#1,rec;rec$
670     PRINT:PRINT"Record added."
680     GOTO 430
1000    PRINT:PRINT"End of parts file
        program."
1010    CLOSE
1020    END
```

This does not presume to be a model program in terms of its error checking, efficiency, or even logic design (note all the Gotos, patently offensive to the initiated). We tried to keep the program simple and straightforward, allowing plenty of room for improvements. One or two things are worth pointing out to help you with your inquiry program. Since each field could be of varying length within certain limits, the backslash character is used to delimit each item. You'll want to strip these out when you retrieve the record. Look up the function Instr; it'll make it easy.

Once you've typed this program in, trying it out can be interesting. Try several values for part number, including some larger ones (greater than thousand, at least). Unless you add records that are sequential, each one will probably trigger a disk access as the appropriate block is written to disk. After adding several, get out of the program by typing Return to the part number and selection prompts and check out the catalog entry on the file. Assuming you used the name MY.PARTS as a file name when you used the create option, the entry will look something like figure 3.

```
TYPE    BLKS     NAME       MODIFIED TIME

TEXT    00007  MY.PARTS     00/00/00 00:00

-------------------------------------------

      CREATED TIME       EOF

      00/00/00 00:00    85376
```

Figure 3.

Look at that EOF value! It seems that you have a huge file until you notice that the Blocks Used column is still pretty small. What SOS has done is report the EOF at the end of the highest record number you used, while allocating only those blocks that it actually needed. Some micros (and some mainframes, for that matter) would require that all the blocks be allocated before any could be written.

Well, have fun until next time. Then we'll try to lighten it up a little as we talk about the mysterious data file type and start using the massive amount of memory in the Apple /// for some really fast indexing schemes. Before this series is over, you should be able to write some pretty hot database programs. Till then, ponder the following: Is it true that disk-based programs are written by BLOCKheads?

Printing Mailing Lists on One Line

Continued from Page 8

Deleting all of the explanatory comments our program looks like this:

```
BEGIN   B
 F*<+>,**A
 F<[=]<<A
 F<@1@<.CRØ>.LM15>.RM95><A
 F<@2@<.LM+25><A
 F<@3@<.LM+30><A
 F<@4@<.LM+16><A
 F<^@5@<.LM+3><A
 F<^@6@<.CR1><A
 PNP
```

Which we saved as "KENS.ONE.LINER". To execute the program, once we have loaded the list we want to print out, into Apple Writer ///, all you have to do is type [P]DO .D1/KENS.ONE.LINER<RETURN>.

In the next issue I will show you how to execute the program faster and build in a prompt requesting the name of the list you want to print out on one line.

Your comments and questions regarding Apple Writer /// Word Processing Language are welcome. In addition if you have a special routine you have developed, send it to us on diskette with lots of explanatory comments.

Recovering deleted or damaged files.

Have you ever deleted a file such as a VisiCalc Model, or an important letter from a diskette then suddenly realized that you needed the deleted files after all. When you delete a file (VisiCalc Model, Appie Writer file, etc.) from a diskette you are actually just deleting it from the diskette directory. As long as you have not added anything to the diskette since your deletion your original file is still intact on the diskette.

Well one of our members, Mike Weathers, has a solution for you. For $ 20.00 per diskette plus postage Mike will recover your data for you.

Mike is also able to recover files which are lost when the directory on a diskette is damaged.

His address is as follows:

Mike Weathers
PO Box 1865
Morganton, N.C.
28655

///

Vanloves Program Writer/Reporter

Vital Information--yes, the same people that brought you the Vanloves Software Directory--have released an interactive database code generator for the Apple ///.

According to Vital Information, Program Writer was developed because of the great demand that the software industry is placing on programmers. As a publisher, they have received many calls from potential computer users requesting non-existing software applications. They feel--and rightly so--that most programmers don't have a thorough enough understanding of the business they're programming for to develop programs that meet the needs of the business community.

Program Writer allows the user to answer a series of logical English questions such as: what does it look like, how long is it, what calculations are needed, what data to accept, what data not to accept? Program/Writer will then write a custom program based on the data input. If this program does what it claims, then it certainly would represent a giant step in the right direction of communicating with computers.

Program Writer can handle records of up to 3,000 characters. The number of records that can be stored is limited only to the physical size of the storage device used (either hard disk or floppies are supported). The display screen is user-definable, this and the complete printer output control allows the user to use pre-printed forms. Calculation and mathematical computation ability is reportedly unlimited. You can write as many programs as you wish and the object codes can be edited. Searches can be instituted by any field or range within a field (for example, you can lookup cities with a population size of between 50,000 and 750,000). Records can be deleted or updated. Files are interactible, meaning that if one record is updated an inventory file is automatically adjusted to reflect the change. Field entries can be protected from erroneous input. Standard database reports are available: selected report print-outs, checks, invoices, mail lists and sales data.

The description of this program fits a powerful database program. The addition of easy English commands should make it easy to learn. Program Writer is written in BASIC which is not really a suitable language for large-record file processing.

Price is $200.00 for Program Writer and $99.00 for Program Reporter.

Apple Support Line

For those of you who encounter many seemingly unsolvable problems there is help! Apple Computer has a support line which can help solve some of those mysteries. At the other end of 408-745-6731 are some very helpful people who can assist you with almost any problem. Of course, computers can be mystifying to even their own makers so don't expect them to solve everything. Problems with programming logic, for instance, are better left to text materials.

open apple gazette

P O BOX 813 SAN FRANCISCO 94101