open apple gazette

Seventh Edition Volume 2, Number 1 July 1983

original apple /// rs

Driving Your Disk: or Shortening the Apple /// Daisy Chain

by Gene Wilson

The Apple /// computer's on-board drive is not meant to be the only means of access to stored data/programs. This is no surprise. It is a fact of life! What then are the alternatives?

"Getting By"

Any user trying to "get by" with a single 140k capacity on-board drive will have to make some major compromises while trying to cope with a machine that can load in various 'system' and program files (with up to a 'current' limit of 256k) that can readily exceed the diskette's total storage capacity. Simple tasks such as copying a diskette can become very frustrating affairs as the user is introduced to frequent 'disk swapping'. In all fairness, some programs will run if a "two-stage boot" is used (which usually involves putting the 'system' onto the first diskette, then swapping to a second 'program' disk). PASCAL? No problem; it simply does not tolerate the single drive environment. Forget it! Clearly then, 'avoidance' is not a solution.

The Elegant Solution

If you are willing to 'double' the purchase price of the machine, the ProFile hard disk is sheer delight. With high speed access to over five million bytes of stored data and programs, the ProFile is certainly worth considering. There is the limitation of disk back-up. How do you channel 5M bytes through that 140k built-in drive? Depending on your 'back-up' needs, this could lead to some "heavy" disk swapping! We'd better keep looking.

The 'Company' Solution

Apple Computer, Inc. is more than happy to provide up to three external drives (daisy-chained, one behind the other), to give the 'system' up to 560k bytes of 'floppy disk' storage. Just think of the pile of units three high, the cost of all that hardware, and 'loading' up to four diskettes into the drives. This isn't the optimum solution either.

The **MICRO-SCI** Solution

(The answer to my problems ..... and maybe yours, too!)

MICRO-SCI sells several disk drive models for the Apple /// computer. Their 'hottest' unit is the **A-143**, which offers 572k of 'floppy' storage. Don Norris, President of the Business Apple Group, provided me with this perfect solution to the 'daisy chain' problem of getting adequate storage on-line. (The Business Apple group sells these units, as a club function, at a substantial savings from the $659.00 retail price).

Features include double-density, double-sided (quad density) operation that boosts the Apple /// to 700k with the first external unit (.D2). With three of these (thru .D4) plugged together, the max. on-line storage becomes a whopping 1.82M bytes.

One external drive gives adequate storage for most applications. PASCAL is completely 'up', requiring NO disk swapping at all!

The drive can also be addressed as .X2, so that standard 140k diskettes can be read (no writing, thank you). This enables two disk copying, or running 'canned' programs requiring two-drive systems.

Initial setup is easy. The required SOS drivers are included. The instruction manual is complete, and gives additional informative tips. One important note here is that the manual says (ever so clearly) that the first 'System Parameter' should be set to ONE Disk /// drive. "The **MICRO-SCI** SOS DRIVER controls all external drives". If that number should read "TWO" (through not reading the instructions the first time thru) then little happens when trying to 'read' from the .D2 diskette.

The system won't find much worth reading, and there will be a list of I/O errors indicating that things aren't going well. Backing up the ProFile? A maximum of nine diskettes is required. This is a far cry from the sheer volume (35 to be exact) of 140k diskettes needed for the same task.

What about "double-sided, double-density diskettes? Expensive? Exotic? Hard to find, perhaps?

The answer was provided by a Business Apple Group member. It seems that most quality diskettes aren't much different on either side. They are usually checked on one surface. Many are certified on that surface, and for an extra price, certification can extend to the back side as well. Certification can be either single, or double density. The bottom line seems to be that just because your diskette hasn't been certified for double-density on both sides doesn't necessarily mean that it won't pass muster. The easy way to find out is to 'FORMAT' and then 'VERIFY' the diskette for full 1120 block operation. If there is a problem, you will be informed!

A word of warning from **MICRO-SCI**. There are two Read/Write heads on the **A-143.** They are offset from one another. If a 35 track diskette (one with a smaller length hole) is used, then one head will 'crash' into the jacket material if the other is fully extended. Words of wisdom from the same friend are that he hasn't seen 35 track openings for a very long time. Nearly all diskette jackets are cut for full 40 track operation. (Just be aware!)

Conclusion

There are a number of solutions to disk storage on the Apple /// computer. Not all are equal in scope or cost. An effective solution is to shorten the 'daisy chain' by using a "high density" disk drive for program and data storage. This solution is not only cost effective, but allows the Apple /// computer to perform a variety of tasks which would normally require a 'hard disk'.

-///-

Any Members Missing Back Issues?

Recently we discovered a bug in our Mail List Manager program that was excluding certain names from our mailings. The bug has since been cured but if you happen to be one of the "Lost Souls", we apologize for any inconvenience this may have caused you. Please drop us a short note informing us of which issues you did not receive and we will make sure you get them. Thank you for your patience.

Learning Pascal

by Raymond Sjerven

In this article, I will review the experiences I had in learning Pascal programming. I had an excellent background in Basic when I started Pascal. I had just finished writing a comprehensive medical office package for the Apple /// in Basic. I'll review the deficiencies I found in Pascal, how I overcame them, and additionally recommend some books for learning Pascal and give some advice on getting started.

I wrote my medical office management package entirely in Basic. It was a big package but it went very smoothly. After the system was up and running, it took about six months to get the bugs out of it. Then, I was left without a programming project. Since Basic's big fault is lack of speed, I decided to rewrite the entire office system in Pascal. Initially, it seemed like a simple enough project. After all, I had already worked out all the system requirements, I was merely translating it into another language. Such naivete was short lived.

Learning Pascal requires a whole lot more than just learning the Pascal version of Basic words. Pascal is an entire system and the entire system must be learned before you can be proficient in it. Also, Pascal is an Edsel. It is designed to make everybody happy and subsequently, it makes nobody happy. Many key processing procedures and functions do not exist in Pascal.

My first three months of working with Pascal were a series of frustrations. I discovered more what Pasal lacked than what Pascal had. I found that you could only "CATALOG" a directory from the System.Filer. There was no "SUB$(" procedure for strings. You could neither get or set a "PREFIX$". It was difficult to "CREATE" a file of the right type without the System.Filer. The "VAL" and "CONV" functions were missing. The Pascal "POS" function would not allow you to start looking anywhere in a string like the Basic "INSTR" function would. "HEX" and "TEN" were missing. There was no "HOME", "INVERSE", "USING", "WINDOW", "RIGHT$(", "VPOS", "HPOS", or "NORMAL".

Also, those Basic words which did have Pascal counterparts were often difficult to find or behaved in a different manner. For example, "ON KBD" in Basic will set up an interrupt whenever the keyboard is pressed. In Pascal, the "KEYPRESS" function has to be repeatedly checked to see if the user has typed anything yet. In Basic, "ON ERR" can be used to mask and detect any error and report it's type. Pascal error checking is limited. "§$IOCHECK+†" and "IORESULT" are only available for disc operations. "LOCK" in Basic has a distinct meaning. The meaning of "LOCK" in Pascal is dependent on how the file was initially opened.

After considerable investigation, it became apparent that Pascal was inadequate for my needs. I looked for a new solution, and I found it. "What is it?", you ask. The surprise answer is, "The Pascal System". To their credit, the creators of Pascal realized that they couldn't devise a language that would make everybody happy. Therefore, they made it possible for programmers to easily extend the Pascal language. Pascal programmers can write new procedures and functions in Assembly language or Pascal and make them readily available to all programs through a System.Library. In effect, Pascal becomes whatever the programmer wants it to be. I need a data processing Pascal. In Assembly language and in Pascal, I wrote the procedures and functions which will enable me to do the programming I want to do.

My first language extension project was a "CATALOG" procedure. I elected to do the procedure as three separate procedures. "OpenDir" would just check the validity of a pathname and return a code. "ReadDir" would return the directory information one line at a time. The data would be returned as a string, along with a code indicating when the directory was out of data. "CloseDir" would clean house. By returning the data as a string, I could use the same procedures to output to the screen, output to a printer, or simply to read the disc. By doing one line at a time, I could also terminate the operation whenever the desired information was located. The procedures were written in a mix of Assembly language and Pascal to take advantage of the power of each.

The next procedure was "SHORTEN", an Assembly language procedure to delete the trailing spaces from a string. The next was "UPPERCASE", an Assembly language procedure to make all the letters in a string upper case.

The next Assembly language procedure was "SUB". This is somewhat comparable to the Basic "SUB$(" but more powerful. You pass to it elements of two bytestreams and an integer value up to 255. It replaces the ASCII codes in the first bytestream with those from the second bytestream. It doesn't take strings per se, but rather the elements of a string. For example:

```
strl:='Jim is tall.';
str2:='Bob';
sub(strl[1],str2[1],3);
     § strl:= 'Bob is tall.'  †
```

"SETLEN" is an Assembly language routine which changes the length of a string. "SETASC" sets a designated number of bytes in a bytestream to a designated ASCII code. For example:

```
setlen(strl,10);
setasc(strl[1],65,10);
     § strl:= 'AAAAAAAAAA' †
```

"RtJustify" is an Assembly language routine which sets a string to a designated length. Excess characters are deleted from the left end of the string. Needed characters are added to the left end of the string as spaces. "RtJustZro" does the same thing, only it fills with zero's on the left. For example:

```
x:=5; str(x,strl);
rtjustzro(strl,3); § ---> strl:='005'   †
rtjustify(strl,5); § ---> strl:='  005' †
rtjustify(strl,1); § ---> strl:='5'     †
```

The prefix procedures are "SETPREFIX", "GETPREFIX", and "PREFIXADD". Here, it is important to point out that Pascal keeps track of a prefix separate from SOS. SOS does a better job than Pascal, so I used it alone. In order to use the SOS prefix from a Pascal program, you must "PREFIXADD" all Pascal file names before they are used. Otherwise, the Pascal prefix will be used instead of the SOS prefix. A pathname which starts with a device name or volume name will not be affected by "PREFIXADD". Prefixing is not the only thing done independently by Pascal and SOS. In most cases, SOS will do the job better though it requires an Assembly language SOS call. Also, SOS calls are not portable to other computers, Pascal operations are.

The data conversion routines I wrote are "VAL" for string to integer, "ValI12" for string to integer[12], "VALR" for string to real, "DOLLAR" for integer[12] to money string, "ValDol" for string (cents) to integer[12], "HEX" for integer to hexidecimal string, and "TEN" for hexidecimal string to integer.

The screen control procedures are "SCREENOFF", "SCREENON", "INVERSE", "NORMAL", "BEEP", "HOME", "WINDOW", "NOSCROLL", "SCROLL", and "CURSOR". Remember, Pascal's screen coordinates are 0 to 79 and 0 to 23 instead of Basic's 1 to 80 and 1 to 24. "WINDOW" takes four integer values, same as Basic. "CURSOR" returns the horizontal and vertical screen positions of the cursor. The Pascal built-in "GOTOXY" will take you anywhere on the screen. Since you cannot change horizontal and vertical positions independently, you may need to "CURSOR" before you "GOTOXY". This concludes my list of Pascal procedures. Next I'll discuss learning Assembly Language and Pascal.

When I purchased my Pascal package for my Apple ///, I was totally unfamiliar with Pascal. I had a heck of a time getting started doing anything. I bought several Pascal reference books to try to get me started. They were all worthless. There isn't anything to know about Pascal that isn't included in the manuals supplied with Apple ///

Pascal. The Apple /// Pascal Manuals are the best available. What finally got me going in Pascal was the examples in the Apple Manuals. I typed them in exactly. They worked. After that I modified them over and over until I got the feel for the system.

Unlike Pascal, the Assembly language information supplied with your Apple /// Pascal system is inadequate. It is however, absolutely essentual. There are no manuals on the 6502 processor in your Apple ///. There are many books available on the standard 6502 processor. The Apple /// Pascal Manuals readily explain the differences between the standard 6502 processor and the Apple /// processor, but they tell you very little about standard 6502 Assembly language. I think I bought every 6502 Assembly language manual written. I found two of them most useful.

"Assembly Language Programming for the Apple II" by Robert Mottola explains Assembly language in the most readable form. None of the examples work on an Apple ///, but it still is a well written, plain language description of Assembly language. Once you're beyond basics, "6502 Assembly Language Programming" by Lance Leventhal is the preferred book. Mr. Leventhal's explanations are well organized and concise. He shows many examples of advanced programming techniques. To first get the feel of Assembly language, use the examples in the Apple/// Pascal Manuals. Once you have the examples working, use the books to figure out what they're doing. When you understand the examples, modify them. Plan on making lots of mistakes. If you blindly do something correctly, you won't learn a thing. The only way to learn anything is to mess up bad, then figure out how you messed up. Be warned, some Pascal and Assembly language routines can destroy all your records in a flash, so back up.

I am deeply into Pascal programming now. I don't think I'll ever have as much fun programming in Pascal as in Basic, but there are compensations. The quality of the final Pascal and Assembly program can be a beautiful thing. I'm done with the struggling and fussing and into highly productive programming. I have my System.Library where I want it and I'm into my medical office package rewrite. The source code for my System.Library is included.

If you have any questions about how it works or why it is written the way it is, please do the following:

First, write your questions on a listing of the source code.
Second, mail the listing to me c/o Business Apple Group, along with a blank diskette. I'll return the source code to you with documentation added which will answer your questions.

Send all inquiries to:
Raymond Sjerven
c/o Business Apple Group
1850 Union Street # 494
San Francisco, CA 94123

- /// -

PASCAL and the MICRO-SCI A143: Or, A Poor Person's Profile

by Richard Lawler

The instructions for Apple /// PASCAL say that an external drive is necessary to use the language system. As it turns out, a single external Disk /// drive is barely adequate to efficiently use the Pascal system for writing extended programs.

The first problem is that the main program, SYSTEM.PASCAL, must remain in the built-in drive almost all the time. If you take it out, you keep getting zapped with messages to put the disk back in the drive and press ALPHA LOCK twice. This, coupled with the problem that the built-in drive cannot hold all the files that are needed for program development, means that your external drive must hold most of the other files like the editor and the compiler in addition to the work and data that you are using. You end up switching disks in and out of the external drive while the built-in drive nurses SYSTEM.PASCAL the whole time. The result is so much disk switching that you excuse yourself from executing any large projects on the system.

A possible solution is to buy a Profile. First shell out $2000. Then you can put Pascal up on the Profile with Catalyst or John Jeppson's patch program from Softalk Magazine (February 1983). (We'll come back to that program in a little bit.) But there is other secondary storage available for the Apple /// besides a hard disk. Probably the best value (at one fourth the price of the Profile) is the **A143** drive from **MICRO-SCI.** It uses standard double-sided disks and holds 560K (that's 1120 blocks) per disk. That's enough room for the entire Pascal system plus 745 free blocks which can hold System Utilities, Quick File or whatever you wish. It plugs right into the back of your Apple /// or Disk ///.

New problems. So you install the **A143** and you have lots more storage but it's still not all that it could be. SYSTEM.PASCAL is still stuck in the built-in drive. The first time the system goes looking for programs like the editor or the compiler, it starts searching in the second drive, then it tries the built-in drive and it then proceeds down the daisy chain until it finds the file. This means that if the **MICRO-SCI** is configured as .D3 the system reads four directories before it finds the program on the new drive.

Now the easiest solution to the latter problem would be to install the **A143** as the second drive in the daisy chain. But there are problems with that idea if you have other Disk ///s: CP/M must have the **A143** last in the chain, and the **A143** cannot be used in Apple][emulation. If you want to use two disks drives in emulation mode then you won't want to have the **A143** second in the chain. Even if you don't consider these to be important setbacks and you do install the **MICRO-SCI** as the second drive, the system disk must still remain in the built-in drive.

But wait: there's John Jeppson's patch program to put Pascal up on the Profile.

Why not use it to put Pascal completely on the **MICRO-SCI A143**? The first reason why not is because the program changes the system volume to the last block device on the System Configuration program's list of drivers. In a standard **A143** setup the driver module includes a driver (.X3 or .X2 or whatever) that allows the **A143** to read standard Apple /// 140K disks. The .Xx driver is after the .Dx driver in the module and they cannot be rearranged. But surprisingly this causes no problems perhaps because the module is loaded by SOS as a single driver. You just have to make sure the module of **MICRO-SCI** drivers is the last block device in the System Configuration list.

The other problem is the dual nature of the **MICRO-SCI A143.** It is addressed and installed like a standard Disk ///, but it has its own driver. (Remember standard Disk ///'s do not have drivers in the SCP. The system normally finds out how many Disk ///'s you have configured from the System parameters.) When using the **MICRO-SCI A143** you have drivers for all external drives, and you configure the System parameters for one disk drive (the built-in).

The Jeppson patch program does not work with three or more daisy chained drives configured this way. It works fine though, if you have only the **MICRO-SCI A143** driver active in the driver file and configure the System parameters for the number of standard drives (this is contrary to the **MICRO-SCI** instructions). This means inactivating the drivers for any Disk ///s before the **A143** on the daisy chain.

Example: If you had one Disk /// and an **A143** in the chain in that order. You would inactivate the **MICRO-SCI** driver for the Disk /// (.D2) and leave only the drivers for the **A143** (.D3 and .X3) active in the module. Then set the System parameter for the number of Disk ///'s to two (for the built-in and the Disk ///).

Once you have got the drivers straightened out, the patch program should work fine. The second of Jeppson's programs in the same Softalk article also works fine with the **A143.** It makes a subdirectory to hold all the Pascal system files. Thus it unclutters the root directory of the system volume. It also causes the system to look first at the new system volume when searching for a program like the editor. This avoids the annoying search through the other directories.

These solutions are not perfect. If you take the system disk out of the new system drive you may get an execution error #10 next time Pascal comes looking for the system disk. It is sometimes a fatal crash. There is little risk of losing data or files because this only happens when switching programs, and Pascal makes sure files are closed. (You can take the system disk out when using the filer.)

Also, a few programs will not work right with the modified Pascal (e.g. The Apple Writer Utilities). So it is a good idea to keep a standard set of Pascal disks handy for some instances. (This is in addition to the back-ups.) Another word of caution. I had some trouble with I/O errors and bad blocks using generic diskettes in the **A143.** I recommend using premium, certified diskettes (such as Verbatim " Datalife") at least for the new system volume.

This new set-up works quite well. It provides two completely free drives and a total of 840K on line. The large size of the **A143** makes the use of subdirectories more practical and thus brings order to volumes that were once quite chaotic. The two free drives allow simultaneous access to applications and data or text without any restrictions imposed by the Pascal system as to what can be removed. Only the 1120 block **A143** drive is partially restricted and it has hundreds of free blocks. With all this convenience you'll have no more excuses not to be productive with the Apple /// Pascal system.

Notes: When running the patch program, be sure to input the number of drives as listed in the System parameters when asked for the number of configured drives.

The patch program needs a slight modification to the two-stage boot error message. Replace the "procedure newMessage" with this version:

```
procedure newMessage:


Place new error message in string
begining at $83B0:
  begin

Move cursor to (0,23) and beep:
  ctrlsl := 'xxxx';
     ctrlsl[1] := chr(26);   ctrlsl[2]
:= chr(0);
     ctrlsl[3] := chr(23);   ctrlsl[4]
:= chr(7);

Clears to end of line:
     ctrls2 := 'x';
     ctrls2[1] := chr(31);

New error message for use with floppy
drive rather that hard disk:

     message := concat (ctrlsl, 'Put
Pascal system disk in device
',devname,        '-Press
RETURN.',ctrls2

     buf[3760] := length (message);

     for i := 1 to length (message) do

     buf[3760 + i] := ord
(message[i]);
     end; (* newMessage *)
```

- /// -

Business Apple Group

Meeting Minutes
for April 20, 1983

The meeting was held at the California State Bar Association building on Franklin St. in San Francisco and brought to order at 7:30 pm by Group President Don Norris.

Don opened the meeting with a discussion of his trip to the recent Applefest held at Anaheim. He said that after talking to many Apple /// owners at this and prior Applefests, it seems that many /// owners feel like 'orphans', in that there is little promotion being done or software being written for the ///.

Don and Group Treasurer Julia Amaral had a booth at the 'fest and sold 30 Gameport III's while signing up 40 new members. They indicated that the /// owners they talked with were especially glad to see a user group and publication for the ///.

Don mentioned several hot new products for the /// that he saw at Anaheim: the **Burtronix** "hi-res" graphics dump; and from **Haba Systems,** a hard disk based telephone time and billing system designed for accountants and lawyers.

The clock chips are still not out yet from Apple, but may be by the "middle or end of May". It seems to be anybody's guess as to when they will be released from Apple. Since installation of the chips is relatively straight forward and they are available from several sources, it was decided to have a clock installation clinic as one of the features next meeting.

Certain 128K ///'s with serial numbers over 100,000 have empty slots that you can plug appropriate RAM chips into and upgrade to 256K for only around $100. Note this is only applicable to SOME ///'s with a serial number over 100,000.

There was a rumor that the 512K machines would be out by the end of the year. SOS is designed to support that amount of memory.

There is a new SOS update, 1.30, which gets rid of some bugs in hi-interrupt operations. In addition to updating the SOS.KERNEL, the diskette also has several new drivers. The format driver makes sure the disk is up to speed before formatting the disk, which gets rid of the 'Volume Not Found' error some users had been getting after formatting diskettes. None of the members had been informed by their dealers of the upgrade.

The CP/M card was mentioned and Woody Liswood was the only club member using it. There is a 2-page list of things to do to get **Wordstar** to run using the card; to be published in the Open Apple Gazette soon.

Jim Linhart brought in a printout he made on the PKASO graphics system. The **Videx** card was mentioned. It features interlacing and gives up to 160 columns. Jim has promised us an article on the PKASO card.

Richard Hart talked about **Catalyst,** a $149 hard-disk system which allows you to use several applications programs simultaneously. He especially liked the documentation. **Terminus** is a new program from Quark that allows terminal connection from within **Word Juggler. Discourse** is a printer spooler for $125. These products are all from Tim Gill at Quark, and a relational database management system will also be available from them around the end of the year.

A **Quickfile** update to 256K will be available soon.

Another Anaheim rumor: one of the largest selling word processors written for the Apple][is being upgraded to the ///. Word is that it will enable the user to work on two files at once through windows.

Although the majority of Apple /// owners are serious business and professional people, there are a significant number who use the emulation diskette to do what Don calls a 'frontal lobotomy' to their ///s and are thereby able to indulge in playing games written for the][. This frivolity has been considerably enhanced by Micro-Sci's **Gameport III,** available from the club at a reduced price. It was mentioned that **Kraft** joysticks are the control device chosen by the serious gamester.

SOS Reference Manuals and Device Driver Writers Manuals are available from dealers.

It was mentioned that there may be a /// upgrade announced in May at NCC; probably featuring enhanced graphics and interlacing. Perhaps this is the upgraded /// referred to by Steve Jobs at the Apple stockholders meeting in January.

Business Apple Group

Meeting Minutes
for May 10, 1983

The meeting was held at the California Bar Association on Franklin St. in San Francisco, and called to order at 7:30 pm by Group President Don Norris.

Don reported that the group sold 9 Gameport III's and 4 Micro-Sci disc drives at the recent Applefest in Boston. He also told of several rumors that he had picked-up on the trip, the most notable being that **Microsoft** is moving their **Multiplan** program to the ///, and that **Visicorp** was dropping development work on the /// to concentrate on their IBM market. The latter was contradicted by opposing rumors. Other rumors: **Pie Writer** from **Hayden Publishing** is moving to the ///. New Versions of **Apple Writer** and **Apple Speller** will be out soon.

Group Vice President Kent Hockabout reported on two software packages that he had tried out; **Vis-Bridge-Sort** and **Vis-Bridge-Report** from **Solutions,** Inc., Box 989, Montpelier, VT 05602. One sorts Visicalc files and the other generates Visicalc reports.

Stuart Forsyth reported that he had gone to the NCC, a large trade show, and found that it is becoming a showcase for products announced in advance. Apple had a large new booth showing **Lisa**, the ///, and //e. Stuart felt that the display was heavily communications oriented. There was a program called **ACCESS 3270** featuring IBM 3270 emulation, and the **Applenet** network was demonstrated, available the last quarter of this year. Apparently no file server is available yet, so initially networking will only be possible between Lisa to Lisa, /// to ///, or //e to //e. Stuart further reported that a new graphics development tool is available for the ///. Primarily for software development companies, it features real interlacing. A records processing service is also available for database development work.

It was mentioned that the new version of **DBMaster** won't run in emulation on the ///.

Stuart mentioned that although **Profile** was designed for the ///, it is being altered for Lisa. Subsequently, some Profiles may be incompatible. If the unit is in for repair the factory will upgrade it.

Don strongly recommends the **Micro-Sci A143** disk drive with 572K for $535, available from the club. In the near future, **Catalyst** will work off a Micro-Sci drive, and they and Quark are working together.

Frances Upton, a consultant, believes that the /// is about to take off, much like VW did in 1954. He pointed to signs in stores that Apple is about to launch a major promotion.

Jim Linhart mentioned that his /// kept on working during a recent brown-out that was measured at 80 volts. He is working with the **PKASO** graphics card, which can print out fotofiles to the Epson printer.

It was reported that Gene Wilson, a long-time Apple][owner (how about serial number 144!) finally got an Apple ///, trading his Corvus for it. Don expects great things from him on the ///.

Richard Lawler reported that Pascal goes up on the A143 Micro-Sci drive using the Jeppson patch. **dBase //** works well with the CP/M card on the ///.

Floppy disks will soon be available from the club at a good price.

Roy Nierenberg, attorney, brought in a program called **SCAT ///**, from Expanding Space Software, 4639 SE 34th Ave., Portland, OR 97202. It allows a master catalog for up to 100 disks on 1 disk, and includes a renumber program. Programmed by Harry M. Sweeney.

Another program for the /// is FYI (For Your Information), now called **Think Tank,** from Living Video Text.

Don mentioned that several people will be coming out with 68000-chip boards for the /// in the Fall. These will increase the speed of the /// by 5 to 10 times.

VersaForm is coming out with an update. It will cost $30.

The **Axlon RAM disk** was mentioned; it speeds up compiling.

The group software librarian is Stan

Guidero. He has Business BASIC Disk #2 available, and is working on #3. Contributions are needed and appreciated.

The Jeppson Patch was brought up again; it works by getting the /// to look somewhere else when it wants to look at the inboard drive.

Business Apple Group

Meeting Minutes
for June 15, 1983

The meeting was held at the State Bar of California headquarters, 555 Franklin St., San Francisco. Group Treasurer and Demo Organizer Julia Amaral announced that **ThinkTank** would be the evening's main demonstration, with **BPI** accounting packages next month and Apple's new **Lisa** machine at the August meeting.

Thirty-seven people were in attendance.

Four monitors were hooked up to one Apple /// for the demos, and it was suggested that a big screen TV would be nice for demos in the future.

The demonstration of **ThinkTank** was presented by David Greene of Living Videotext, Inc., 450 San Antonio Road, Suite 56, Palo Alto, CA 94306. Phone(415) 857-0511. The program was written by group member Jonathan Llewellyn and Dave Winer. It seems that ThinkTank is an "idea processor" which allows one to generate, organize, store, and retrieve ideas and information by using a dynamic outline structure.

Examples of the way it might be used are as a notebook, address book, appointment calendar, card file, secretary, or administrative assistant. It can be used for database management on a hard disk. It has its own BIOS and is written largely in Apple Pascal with machine-language modules. It is claimed to be ideally suited for the front-end work of writing, although it is not as good as a dedicated word-processor (it doesn't do formatting). Club member Stuart Forsyth has had the program for awhile (it was originally called FYI), and "uses it quite a bit" for such things as meeting planning and as a calendar.

It was announced that there will be a Board meeting on Monday, July 11 at 7:30 at the Bar assaociation.

Julia Amaral was commended for the work she's done on coordinating future demos, and Richard Lawler volunteered to be the Pascal program librarian. Other club needs are for software reviewers, article writers, a software list keeper, editorial contributions, a nationwide article solicitor, and an advertising coordinator.

Group President Don Norris announced that a new software list for the /// will soon be available. He then introduced an employee from Apple who observed that the Third Wave promo is starting to take hold at Apple, pointing out that there are over 400 software packages for the ///. It was also noted that a lot of people who develop on the /// don't communicate or distribute; that some of the highest quality program writing is being done on the ///; and that IBM has belatedly picked up the concepts of drivers and pathnames.

It was confidentially rumored that a new thing called "Driver's Aid" will be available soon for easy installation of drivers on diskettes. We also learned that the ///+ will be out in August or September. Carolyn also mentioned that she thinks **Catalyst** from Quark is an outstanding program.

Don pointed out that the **Gameport III**, available from the club, will work with all games including Broderbund's popular Choplifter. Frances Upton noted that the card may interfere with Word Juggler operation, however. Micro-Sci was praised for its support.

Quark is now advertising **Terminus**, an $89 communications package which operates from within Word Juggler.

Richard Hart asked the Apple representative why Apple dealers don't seem to know anything about the ///, and suggested that an organized response to the marketing manager for the /// might bring some results.

Group member and consultant Woody Liswood demonstrated a program called **Selector**, which runs under CP/M and therefore requires a CP/M card to run on the ///. It is available from Micro-Ap, Inc., 7033 Village Parkway, Suite 206, Dublin, CA 94566, phone 415-828-6697.

Selector is a fully relational database system, in a class with dBase][and Condor which also run under CP/M. For comparison purposes, consider Quickfile ($100) from Apple, which runs under SOS; dBase][(discounted to $400); and Selector ($495). dBase is the best seller, and is the standard by which to compare the others.

The programs are used for custom installation of accounting, inventory, and other related business needs. Woody recommends **Selector** over dBase for two main reasons: Selector is nine times faster than dBase, and Selector can have six files open at once while dBase can only have two. Quickfile can only have one file open at a time, which limits its usefullness considerably. You can have 89-step modules with Selector. Woody also pointed out that a **ProFile** hard disk is essential for operation of the program.

Business Apple Group

Meeting Minutes
for July 20, 1983

The meeting was held at the California Bar Association on Franklin St. in San Francisco, and started early with a demonstration of Dean Witter's new stock market software.

The new Apple /// magazine ON THREE was mentioned, and it seems to be largely concerned with BASIC and Pascal programming articles and software reviews. Apple recently included a promo for the magazine in a mailing to all /// owners (of record).

Francis Upton of Computerland Oakland announced that dealers have received instructions for installing our long-awaited clock chips.

Group Treasurer and Demo Organizer Julia Amaral then introduced Pete Levy, product marketing manager at Apple, who was invited to show us the new **BPI** accounting package.

It was emphasized that the package is a new product, as some members expressed dissatisfaction with the old one which ran on the][.

Mr. Levy opened the presentation by pointing out that accounting products are second in complexity only to operating systems.

The BPI system requires a hard disk and a 256K machine.

There are four basic accounting system functions; general ledger, accounts receivable, accounts payable, and payroll. Secondary functions might be job costing and inventory control. The BPI system has the ability to keep names, addresses and account history. Pete feels that this system will be contributing to making the next few months super exciting for the Apple ///.

Promo plans for the ///-BPI package were revealed, inclucing a national print ad campaign. The four basic modules will be available in September, with job costing and inventory control becoming available in January.

Pete then presented a slide show and discussion in detail of the various components of the system, accompanied with extensive literature.

Stuart Forsyth asked if a point of sale module was under development, and it apparently is not. Business graphics will not interface with the system. Stuart also gave his definition of a small business as a company that can handle their accounting function on one keyboard. He suggested that if your company needs more than one keyboard to go to **Unix** on the **Lisa** machine.

The BPI package works with **Discourse** and **Catalyst.**

The competition for BPI is **State of the Art** and **Great Plains.** Compared to BPI, they are much more customized and therefore present additional time in set-up.

Pete concluded the presentation by passing out his business card to everyone present and suggested that anyone with questions call his office. Demo diskettes of the program may be available.

Random notes:
Apple File /// is coming. Data files cannot be transferred from /// to Lisa.
The ///+ is late.

Respectfully submitted,

Charles Coles
Secretary

Business BASIC Disk No. 2

Business Apple Group
Apple /// Software Division

DESIGN...........This program is a Character Generator written by John Jeppson and appeared in Softalk magazine. The version we offer was typed in and de-bugged by Steve Bowles. You can create and save your own character set. Programs that are working parts of DESIGN are:

NEWDOWNLOAD
NEWSET
READCRT.INV
DOWNLOAD.INV
REQUEST.INV
BACKWARD

NEWSET and BACKWARD are font files that can be read by DESIGN.

DESIGNLETTER.....is a text file letter by Steve Bowles giving further details on the program. This may be read using TEXT.FILE.SCRLL.

TEXT.FILE.SCRLL..This program will allow you to use the UP, DOWN, LEFT and RIGHT ARROW Keys to view a text file. This program originally appeared in Softalk magazine and was written by Taylor Pohlman. It was typed in and improved by Mike Kramer.

HEX.DEC.CONVERT..A program that will convert HEX code to DECIMAL code.

CARD.CONVERTER...A Utility that works with Apple Computer's QUICKFILE ///.

CARD.COL.SPRED...Works with QUICKFILE ///.

HEX.ASCII.DUMP...Will take a text file and create another text file but will have ASCII code included.

LIFE.............Also a program by John Jeppson that appeared in Softalk. It is a graphic program that requires BGRAF.INV to be on the same disk.

Business BASIC Disk No. 3

Business Apple Group
Apple /// Software Division

HELLO.............The Hello program has some good programming techniques which you may find useful.

DOC.READER........This is a file reader that will allow you to read this file which was written with Apple Writer ///. You can use the up and down arrow keys to view the text. Use the ESCAPE key to exit and catalog this disk.

DISK.DOC..........The text file containing this information. This file is written with Apple Writer ///.

CATALOG.MENU......This program converts the directory to a program menu. It displays up to 50 files, 13 per page, which can be run with a single key. Pages are turned with the SPACE. You can return to Business BASIC with ESCAPE. You can also print the catalog with <P> (Control-P). Be sure your disk contains READCRT.INV.

LABELER.1.5.X.4...Tired of wondering which files are on your disk? This program will print all of your filenames onto Avery 1.5-inch X 4.0-inch labels. You are also prompted to give the disk a DISK NAME (CR will leave this blank). You are then given the option to print to <S>creen or <P>rinter. The program is set up for an Epson MX-100 with Graphtrax Plus in lines 605-609 which can be changed to meet the needs of the user.

LABELER.3.X.5.....This is similar to LABELER.1.5.X.4 except it is for Avery 3-inch X 5-inch labels which look nice on the disk en- velope. It also has an additional option <F>ree Text which is a simple text file maker to write your on proseon the label.

AUTO.INVOICE......Will format Visicalc(TM) files for output to invoice forms Example files are included. They are:

FOBLANI283.VC
FOBKAUAI.46.VC
FOB.SHEET.VC
DEMO.DIF

MAKE.CUST.LIST...This program creates the file called CUSTOMERLIST.

CASE.CONVERTER...Will convert text files to all upper case or all lower case.

COMPARE..........Will compare two programs or text files for differences. (You will use CAPTURE.EXEC when operating COMPARE)

SORT.DEMO.BASIC..Demonstrates a typical sort routine as used in Business BASIC.

READ.CAT.DEMO....Will read text files and output it to the screen.

WAR.AND.PEACE....War and Peace, written by a college student, Chris Handy, is a game of international conflict which places you in charge of world-wide affairs by assigning you the role of the President of the United States. Your goal is to act in whatever manner necessary to sustain world wide harmony in the near future as you match wits with the computer-guided Soviet Union. See if you can save the world. (Your own version of "War Games", Global Thermonuclear War)

Business BASIC Disk .004

Business Apple Group
Apple /// Software Division

Donated by Dr. Raymond Sjerven

This is a letter writer written in Business BASIC, enabling you to create a document of up to 160 characters wide and 100 lines long. After working with word processors written in Assembly Language, such as Word Juggler and Apple Writer, you will find it is noticeably SLOWER. But it was written to be incorporated as part of an overall integrated medical office management package and served a very useful purpose for Dr. Sjerven, since at the time there were no medical software programs available. Dr. Sjerven has graciously donated this to the organization to help others out there get the most out of their ///s and learn just what you can do with Business BASIC.

Given the speed limitation, it is a straight forward menu driven program which is easy to use. The menu's are similiar to those of the System Utilities Program, ie: an inverse bar is over one of the menu selections; to choose that option, merely press <RETURN>. Other menu options can be selected by moving the inverse bar with the up and down arrows to the item of your choice and then pressing <RETURN>.

It is good programming practice to follow a standard menu selection, such as the one you see when using System Utilities Program. It is easier for the end user to move from one function of the program to another.

The Mailing List Program is also very user friendly and easy to follow with out documentation.

Adding this program to your library will provide you with many useable programming techiques.

- /// -

Your favorite Business BASIC programs are needed for inclusion in future Business BASIC diskettes featured on our Public Domain Library. If your program is used on a future BASIC diskette, you will receive a free Business
BASIC diskette.

Order your **Business BASIC** diskettes now for only $10.00 each, plus $2.50 postage/handling. Foreign orders should add $2.00, Canadian $1.00. Foreign orders should be in US Funds drawn on a US bank or an International Postal Money Order payable in US funds.

Save time formatting, copying and labeling and order a back-up at the same time for only $5.00.

Order from:
Business Apple Group
1850 Union Street,#494
San Francisco,CA.94123

Beginning Business BASIC
Lesson 3

by Stan Guidero

Before starting this session, I have some corrections to make to our last lesson. In our first program example which we named ADDIT from Volume 1, Number 6, I listed a change adding lines 10 and 20. Line 20 should have been line number 35. Also, the REM statements didn't quite come out the way I wanted due to formatting problems with printing. The corrected Trip Cost Program will be reprinted at the end of this lesson.

To continue with our last lesson, we were discussing decision making with the IF-THEN-ELSE statement. This statement is one of the most powerful available to you in BASIC. Last issue I stated that the ELSE is used mainly for clarity and is optional. Although that is true, the ELSE can be used to extend the power of the IF-THEN statement, as in the example that follows. We can take the two statements:

```
   100  IF S=1  THEN GOTO 500
   110  IF S<>1 THEN GOTO 600
```

Can be written:

```
   100  IF S=1 THEN GOTO 500 :ELSE GOTO 600
              OR
   100  IF S=1 THEN 500 :ELSE 600
```

which is not only shorter but clearer.

Another use of the IF-THEN-ELSE statement is the look-up table. The following program will demonstrate this. Have you got Business BASIC up and running? You don't! Come on now. Turn off the TV and go to the Apple ///. Place your seat in the seat of the chair and start up Business BASIC.

(NOTE: If you're reading this on your way to work or home, put this away or read another article until you get to your Apple ///.)

If you're at the Apple /// , turn on the computer and..............hummm!

I think we have a pseudo code here. What's a "pseudo code"? Using a pseudo code we can outline the start of a program. If I were to write a pseudo code based on the above directions, it might look like this:

```
   IF BASIC = running THEN continue ELSE
   IF at home = NO THEN go home ELSE
   IF commuting GOTO home
   IF TV = on
   THEN turn TV off goto Apple ///
   IF at Apple ///
   THEN turn on computer with
   Business BASIC
   CONTINUE -> Apple /// running Business
   BASIC
```

Well, as you can see, just about any decision making can be handled by this statement. Now that you have Business BASIC up and running here is the program:

```
   )NEW
   10  REM   * A discount table program *
   20  HOME
   30  INPUT "Customer discount
              (A,B,C OR D) "; DISCOUNT$
   40  INPUT "What is the retail
               price "; PRICE
   50  REM -------LOOKUP TABLE-------
   60  IF DISCOUNT$ = "A" THEN CREDIT = .05
   70  IF DISCOUNT$ = "B" THEN CREDIT = .1
   80  IF DISCOUNT$ = "C" THEN CREDIT = .15
   90  IF DISCOUNT$ = "D" THEN CREDIT = .2
   110  SALESPRICE = PRICE-(PRICE * CREDIT)
   130  PRINT "The discount
                price is ";SALESPRICE
   999  END
   )RUN
```

This is a very simple and straightforward look-up table. Lines 30 and 40 get the information needed for calculation. Lines 60 thru 90 are the actual look-up table. When DISCOUNT$ equals your input, then CREDIT will equal the amount on that line. If you input the letter B for customer discount, the program will search up the look-up table until it reaches line 70. Finding it true will make CREDIT equal .1 for a 10% discount. The program continues to line 110 where the actual calculation takes place, and finally to line 130 were the results are printed to the screen. Let's save this program for future use:

```
   )SAVE DISCOUNT
```

The look-up table has many uses. It could be used as a program menu. An example follows.

```
   )NEW
   10  REM * A MENU PROGRAM *
   20  HOME
   30  PRINT TAB(17);"** MENU **"
   40  PRINT
```

```
50 PRINT TAB(25);"1) Section One"
60 PRINT TAB(25);"2) Section Two"
70 PRINT TAB(25);"3) Section Three"
80 PRINT TAB(25);"4) Section Four"
90 PRINT TAB(25);"5) QUIT"
100  PRINT
110  INPUT "   Make a selection ";SELECT
120  IF SELECT = "1" THEN 200
130  IF SELECT = "2" THEN 300
140  IF SELECT = "3" THEN 400
150  IF SELECT = "4" THEN 500
160  IF SELECT = "5" THEN 999
200  PRINT "Here's One!"
210  GOTO 999
300  PRINT "Here's Two!"
310  GOTO 999
400  PRINT "Here's Three!"
410  GOTO 999
500  PRINT "Here's Four!"
510  GOTO 999
999  PRINT "GOOD BY!!":END
)RUN
```

Lines 30 thru 110 prompt you for your choice and lines 120 to 160 are the look-up table for your selection (SELECT is the variable used). Lines 200, 300, 400 and 500 are dummy lines representing the selected section which could easily be program sections. This type of look-up table is sometimes called a CASE Statement. The IF-THEN-ELSE Statement gives you two ways to go were the CASE type can give you several choices.

So that you're not confused, there is no statement in BASIC that uses the word CASE. However, some other languages like Pascal do use the word CASE. If you had many selection paths to use, the IF-THEN statement could get rather lengthy. Fear not, for help is at hand in the guise of the ON-GOTO statement. You can replace lines 120 thru 160 with one statement. But first, let's save our existing program. Type:

```
)SAVE MENU1
```

Now:

```
)DEL 120-160
```

This will remove lines 120 thru 160.

```
)LIST
```

To make sure the computer did its job, next type:

```
150 ON SELECT GOTO 200,300,400,500,999
```

```
)LIST
```

```
)RUN
```

The program should work exactly the same. The ON-GOTO looks for the value of the variable SELECT and moves the pointer which starts at the first line number in the statement. A "one" would leave it at the first number. A "three" would move it to the third line number in the statement and so on. This statement can be used with GOSUB instead of GOTO. A GOSUB is similar to a GOTO except you must use the word RETURN to return back to the next line. If we had used ON SELECT GOSUB 100, 200 etc.,we would use RETURN in place of the GOTO statement in the dummy lines.

Thus line 200 PRINT "Here's One!" would be followed by 210 RETURN and the same for the rest of the statements thru line 510. One word of note: the RETURN sends the computer back to the line following the line with the last GOSUB. In our program this is unfortunately back to line number 200. At this point the computer will print "Here's One!" and then finding a RETURN without it's corresponding GOSUB, it will display an error message. What we have to do is to add a new line. Lets add:

```
190  GOTO 20
```

Now when the computer RETURNS to the next line, it will be directed to line 20 which displays the menu again. Of course line 999 remains the same and ends the program if QUIT is selected. Well, that's all for now. Next issue we will continue with error-trapping using the IF-THEN statement. Happy programming.

Here is the corrected Trip Cost Program as promised:

```
  5 HOME
 10 PRINT "TRIP COST"
 20 PRINT
 30 PRINT "Miles","MPG","Time","Fuel
           Price","Total"
 40 PRINT "Traveled","","for trip",
          "per.gal.","Expense"
 50 PRINT "--------","---","--------",
          "--------","-------"
100  REM -----------------------------
110  REM    Read values from data
120  REM -----------------------------
125 :
130  READ  MILES,MPG,TYME$,FUELPRICE
140 :
200  REM -----------------------------
210  REM      Do calculations
220  REM -----------------------------
```

```
225 :
230  TOTAL = MILES / MPG * FUELPRICE
235 :
300  REM -----------------------------
310  REM      Print it out
320  REM -----------------------------
325 :
330  PRINT MILES , MPG , TYME$ ,
            FUELPRICE , TOTAL
335 :
400  REM -----------------------------
410  REM        Do it again
420  REM -----------------------------
425 :
430  GOTO 130
440 :
500  REM -----------------------------
510  REM       Here's the data
520  REM -----------------------------
525 :
530  DATA  125,25,"1:30min",1.33
531  DATA  245,25,"2:45min",1.33
532  DATA  578,27,"6:50min",1.33
533  DATA  35,23,"45min",1.34
999  END
```

-///-

STEMS AND SEEDS:

Ed. Note: Well, the votes are in and it's unanimous. The title for this section will be STEMS AND SEEDS. Please send us your items to be included among these other valuable bits of info.

APPLE WRITER /// And CONTROL-O:

When using APPLE WRITER ///, sometimes after attempting the Glossary function (Control-O), only one line of information from the Catalog will appear on the screen. I have found a way to reset the command back to its normal function. Press Control-O, chose option 1 and then give a letter like "E" for the drive number. You will get an error message. Press return and the system returns to normal without erasing your text. You can then Catalog and see the full catalog instead of only one line.

Gaston Savoie
St. Bruno, Quebec
Canada

- /// -

Exploring Business Basic - Part Six

By Taylor Pohlman
Reprinted from Softalk Magzine

Last episode covered a mixed bag of topics and ended with a promise to cover some parts of the new "request" invokable module and techniques on using "print using." Fear not, all that and more is covered herein, including some tips on long integer decimal arithmetic. But first, a few digressions based on comments some of you made on previous articles.

Digression Number One. One of the first articles asserted that random record files were limited to 32,767 records, the maximum positive integer value. In fact, there is no particular limit in SOS on which this Basic limit is based. Basic even allows a real number to be used as a record number, but because Basic uses an integer type internally to keep track of the record number, the value still cannot exceed 32,767. The actual position in the file is determined by multiplying this record number by the record size assigned when the file was originally created (default is 512 bytes). This 32,767 limit on record numbers does not hold for Pascal. Now that the Profile hard disk is available, some thought is being given to removing this restriction. Speak up if it's been a problem for you.

Digression Number Two. As demonstrated last time, the "get#" statement in Basic can be used to read the exact contents of most files on the Apple III, one byte at a time. We even created a special formatted dump program to investigate the contents of Basic "data" files. Some types of disk files cannot be opened by Basic, however. Most notably, these include Pascal "code" and "data" files. If you need to examine the contents of those files from Basic, you can do so by using the Pascal Filer to change the file type to ASCII, which Basic knows as the "text" file. You Pascal prorammers will enjoy Basic once you try it!

There is another file type which is very interesting to examine, and Basic will allow you to open it directly. Those are the "Catalog" or "Subdirectory" files, which you create from Basic or the Utilities program. The subdirectory capability of SOS is one of its most powerful features. If you aren't using subdirectories to group your files and programs logically, you might want to read the relevant sections of the Basic manual and the Apple III "Owner's Guide." One problem with files in Basic, however, is that it is difficult to discover from a running program whether or not a given file or program already exists. There are some ways using "on err" to work up a solution to this problem, but nothing very tidy. However, being able to open and read a directory or subdirectory allows us to check on everything before opening a file or chaining to another program.

Those of you who read last month's article know about our handy-dandy file dump program using "get#." Let's pick a typical subdirectory named "mysub" containing the files "myprogram" and "directorydump." Using the formatted dump program from last time, the file contents look something like Figure 1.

For those of you who did not read last month's column, this may look bizarre, but it's really easy. Remember that this is a byte-by-byte image of the file. The numbers to the left (like 0000-001F) are the byte numbers in hexadecimal of that particular row. Each row contains thirty-two bytes. The top row in each pair is the actual hex contents of the file, and the next row is the ASCII equivalent characters. If the byte is a nonprinting character, it is reresented by a period. This is all fine, but you will immediately protest that other than being able to spot the subdirectory name and the file names, the printout is a big mystery. Business Basic to the rescue! It turns out that Basic is knowledgeable of the contents of directory files, so that when you open a directory or subdirectory file, Basic will automatically format the contents for you, just as it does in the "catalog" command. The following simple program will illustrate, on the same subdirectory we just looked at:

```
1   INPUT"Directory to dump: ";a$
10    OPEN#1,a$
15    ON EOF#1 GOTO 60
20    INPUT#1;a$
30    PRINT LEN(a$)":"a$
50    GOTO 20
60    CLOSE
70    END
```

The only thing unusual here is that we arranged to print the length of each string that is read, to check for any special formatting. The output looks like Figure 2.

Since all the columns are in very predictable places, it is possible to extract the information desired easily by judicious use of the "mid$" function. Also, since this is a subdirectory, there is no line showing blocks free and blocks in use. Try using this program on a volume directory. The last string read from the file will contain this information, very useful if you want to check for imminent disk full errors. Also, since the volume directory lists all the subdirectories (labeled CAT in the file type column), it is possible to get a full list of all the files on a volume by successively reading the individual subdirectory files. Another treat for the esoteric members of the audience is to compare the information in the hex dump with the formatted output to discover where and how SOS hides all the information about files.

New Stuff. As promised last time, we'll now go briefly into one of the most powerful new capabilities of Business Basic, the "request" invokable module. Normally, all access to SOS files is done through the "input," "print," "read," "write," and "get" statements of Basic. Basic interprets your desires and performs operations called SOS "calls" to do the actual work of reading and writing to physical devices. There are times, however, when the programmer needs direct access to the information which SOS has about files, and other times when certain status and control information needs to be interrogated or set. More information about what this information consists of for a particular driver can be found in the appropriate reference manual for that driver.

Of greatest interest to us now, however, is the ability to use SOS directly to read and write data to files. A single SOS "fwrite" command can transfer up to 64K bytes of data to a file. Normally Basic allows writing only one variable at a time, and although it is possible to put more than one value in a single "print" or "input" statement, there are real limits on the amount of data which can be transferred at one time. This generally means that arrays of data get written using "for-next" loops--adequate, but hardly a speed-burner.

To help solve this problem in situations where performance is at a premium, the "request" module contains two procedures: "filread" and "filwrite." They are documented in the "request.doc" file on the Basic disk, but for reference, here are the formats:

```
   PERFORM    FILREAD(%filnum,@array$,%num-
      bytes,@count)
   PERFORM    FILWRITE(%filnum,@array$,%num-
      bytes)
```

"Filnum" refers to the file number you used in the "open" statement for the file to be read or written. It can be any file which Basic is allowed to open. This includes device files like .console as well as disk-based "text" and "data" files. The percent symbol in front of the "filnum" indicates that you should either use an integer variable, or put the % in front of any constant you use, to insure that an integer value is passed to the procedure. "Array$" refers to a string variable which contains the name of the array which you wish to read or write. The @ character on the front of the string variable name instructs Basic to pass the memory address of the string, not the actual contents of the string itself. The invokable module is responsible for finding out what array name is in the string, and then locating the array in memory. The "numbytes" parameter tells the procedure how many bytes are to be read or written from the array. In the "filread" procedure, the extra parameter "count" allows the procedure to pass back information about how many bytes were actually read, in case an "eof" or other event prevented the reading of the full amount of data specified. It must be an integer variable.

One note is important here. These procedures read and write the exact contents of arrays. In the case of disk files, there is no way to read this data back once it is written, except by using the "filread" procedure. That is, if you write an integer array to a "data" file, no type bytes are placed in the file, just the binary integer values, one after the other. The same is true for "text" files. Normal writes to "text" files convert the binary internal format to ASCII character format. If you write a "text" file using "filwrite," the exact binary data is written. You can position the file pointer using random access statements, but once a "filwrite" starts, it does not respect record boundaries. Great care must be taken if you have any ideas about mixing this kind of data with the normal contents of "text" and "data" files. A good approach is to use record 0 of the file to document the use of "filread" and "filwrite" within an ordinary file by putting information there about the types of arrays, their location within the file, their length, and so forth.

Now that we've documented how it works, let's look at an example which will demonstrate how it can improve the performance of your programs.

The following program represents a benchmark of the time it takes to write a real and an integer array to a data file:

```
   10   DIM realarray(10,100),intarray%(10,
        100)
   20   OPEN#1,"test.request"
   30   REM fill arrays with random data
   40   FOR i=1 TO 10
   50   FOR j=1 to 100
   60     val=RND(1)*30000:valint%=INT(val)
   70     realarray(i,j)=val:intar-
          ray%(i,j)=valint%
   80       NEXT j,i
   90   PRINT"Arrays filled."
   100  PRINT"Writing  real  array  with
        FOR-NEXT."
   110  PRINT"Start time: ";TIME$;
   120  FOR i=1 TO 10:FOR j=1 TO 100
   130    WRITE#1;realarray(i,j)
   140    NEXT j,i
   150  PRINT" Stop time: "; TIME$
   160  PRINT"Writing  integer  array  with

        FOR-NEXT."
   170  PRINT"Start time: "; TIME$;
   180  FOR i=1 TO 10:FOR j=1 TO 100
   190    WRITE#1;intarray%(i,j)
   200    NEXT j,i
   210  PRINT" Stop time: "; TIME$
   220  CLOSE
   230  END
```

As you can see, this is a relatively straightforward program that writes a 1000-element real array and a 1000-element integer array to disk. Apologies to those of you without clock chips. If you run this program, the timings should look something like this:

```
   )RUN
   Arrays filled.
   Writing realarray with FOR-NEXT.
   Start time: 13:37:42 Stop Time: 13:38:17
   Writing integer array with FOR-NEXT.
   Start time: 13:38:17 Stop time: 13:38:38
```

All this adds up to about thirty-five seconds to write the real array, and thirty-one seconds to write the integer array. A great deal of this time is spent in the "for-next" loop and in writing each element separately. Now let's look at the same program using "filwrite":

```
10   DIM realarray(10,100),intarray%(10,
     100)
20   OPEN#1,"test.request"
25   INVOKE".d1/request.inv"
30   REM fill arrays with random data
40   FOR i=1 TO 10
50     FOR j=1 TO 100
60       val=RND(1)*30000;valint%=INT(val)
70       realarray(i,j)=val:intar-
         ray%(i,j)=valint%
80        NEXT j,i
90   PRINT"Arrays filled."
95   array$="realarray"
100  PRINT"Writing real array with FIL-
     WRITE"
110  PRINT"Start time: "; TIME$;
120  PERFORM filwrite(%1,@array$,%4000)
150  PRINT" Stop time: "; TIME$
160  PRINT"Writing  integer  array  with

     FIL-WRITE"
165  array$="intarray%"
170  PRINT"Start time: "; TIME$;
180  PERFORM filwrite(%1,array$,%2000)
210  PRINT" Stop time: "; TIME$
220  CLOSE
230  END
```

Notice in the filwrite "perform" statements, that %1 was used to denote the fact that we wanted to write to file number 1, and the string "array$" first contained the array name "realarray" and then "intarray%." Also, a length of 4000 was used in the case of the real array (1000 elements at 4 bytes each) and 2000 in the case of the integer array (1000 elements at 2 bytes each). The result when this version is run is quite dramatic:

```
)RUN
Arrays filled.
Writing real array with FILWRITE
Start time: 13:54:45 Stop time: 13:54:48
Writing integer array with FILWRITE
Start time: 13:54:48 Stop time: 13:54:49
```

That's right! Approximately three seconds were required for the real array, and only one second for the integer array, between ten and twenty times faster than the previous example. Remember, though, that data written with this technique is readable only with a similar "filread" statement, and if you ever lose track of the way in which it was written, it's tough toenails. Even with those minor difficulties, you're sure to find lots of good uses for this new invokable module.

New Stuff-Part Two. For several months now you've been promised some information about the "print using" capabilities of Business Basic. Rather than go into detail about every little feature, we'll take a quick look at the main features, and then examine a program that shows off some of the power of "print using." We'll also explore the use of the long integer data type for financial accounting applications. That's a lot to stuff for one section, but here goes:

Like most "print using" implementations in various dialects of Basic, Business Basic permits the printing of a list of variables according to a format described in an "image" statement. In fact, if you have programs in Microsoft Basic, Cbasic or most others with simple "image" statements, they should convert readily. It is in the extensions to these simple capabilities where Business Basic really starts to shine. The standard format is, as was said, like the following:

```
10   PRINT USING 20; first$,firstnum,
      secondnum%
20   IMAGE AAAAAAAAAAAAAAA,XXX,#####.
      ##,XXX,#####
```

In the "image" statement, A reserves a space for one alphabetic character, X inserts a blank space, # reserves a space for one numeric digit, and . tells Basic where to align and print the decimal point in a numeric field. Therefore, the example in line 20 is interpreted as follows:

"Print the string variable "first$" in the first fifteen positions of the output record, skip three spaces, then print the real variable "firstnum" with five digits to the left of the decimal point, and two decimal places to the right. Then skip another three spaces and print the integer variable "secondnum%" right justified in a five-digit field."

Assuming the values "My test string" for "first$," 123.443 for "firstnum," and -2345 for "secondnum%," the output would look like this:

```
My test string 123.44 -2345
```

Other questions, like what happens when the number or string is too big to fit, are best left to a careful reading of the Basic reference manual. Now the fun begins. Business Basic allows considerable flexibility in the way the simple example

```
0000-001F  00000000E54D5953554200000000000000000000760000000000000009DA3060F

           . . . . e M Y S U B . . . . . . . . . . v . . . . . . . . # . .

0020-003F  000000270D020009000227194D5950524F4752414D0000000000000982000100

           .  . . ' . . . . . . . ' . M Y P R O G R A M

0040-005F  A301009DA3070F0000E300029DA3070F81002D4449524543544F525944554D50

           # . . . # . . . . c . . . # . . . . . - D I R E C T O R Y D U M P

0060-007F  000004840006007609009DA3090F0000E300029DA3090F810000000000000000

           . . . . . . . v . . . # . . . . c . . . # . . . . . . . . . . . .
```

(Figure 1.)

```
68:  MYSUB       (12/29/81)  VO

68:

68:  TYPE   BLKS   NAME            MODIFIED  TIME  CREATED  TIME  EOF
65:  BASIC  00001  MYPROGRAM       12/29/81  15:07 12/29/81 15:07 419
66:  TEXT   00006  DIRECTORYDUMP   12/29/81  15:11 12/29/81 15:09 2422
68:
```

(Figure 2.)

Literal Spec

X	prints a space
/	prints a carriage return
"any text"	inserts literal strings in the output

Digit Spec

#	Reserves one digit, leading zeros are suppressed
&	Reserves one digit or comma. Commas are inserted every 3 digits
Z	Reserves one digit, leading zeros are printed

Special Numeric Specs

+	Reserves position for a sign
-	Prints sign only if negative (default)
++	Prints "floating sign" in rightmost unused position
--	Prints "floating sign" only if negative
$	Reserves position for dollar sign ("$")
$$	Prints "floating dollar sign"
**	Fills leading spaces with asterisks
E	Prints the number in scientific or engineering notation

String Specs

A	Prints string left-justified in the field
C	Prints the string centered in the field
R	Prints the string right-justified in the field

(Figure 3.)

above can be expressed. For one thing, it can be simplified by placing repeat factors on the specification characters, like this:

```
20   IMAGE 15A,3X,5#.2#,3X,5#
```

Another feature is that the "image" string can be a string value replacing the line number reference in the "print using" statement. The following are equivalent:

```
10   PRINT USING
     "15A,3X,5#.2#,3X,5#";first$,firstnum
      second-num%
10   format$="15A,3X,5#.2#,3X,5#"
20   PRINT USING format$;first$,firstnum
      second-num%
```

It is this last variation, and the power it gives us to change the format under program control, that we will explore in depth a little later.

So far we have covered the X specification, called a "literal" spec, the A spec, called a "string" spec, and the # spec, called a "digit" spec. Others available are shown in Figure 3.

As you can see, these options give the programmer quite a bit of flexibility in outputting information, especially in business and scientific applications. What gives even greater flexibility is the fact that "print using" works with files, by using the "print using#n" form of the statement, and even works with random access text files by substituting "print using#n,rec."

One other feature of "print using" is important to mention. Many business programmers, especially in accounting applications, must use integer arithmetic to ensure "penny accuracy"--no round-off errors from floating-point calculations. Ordinary Basics hamper this effort, however, because "print using" cannot insert decimal points in integer values. Business Basic has a special function, used only in "print using" output lists, to solve this problem. The function is called "scale," and can be used with any numeric value to apply a relative power of ten (decimal point shift) to the number being printed. The format looks like this:

```
SCALE(scalefactor,numericvariable)
```

For example, the following:

```
10   longnum&=12345678
20   PRINT USING "7#.2#";SCALE(-2,long-
      num&)
```

would result in the output:

```
123456.78
```

To illustrate the use of these features in business applications, the following program will be used. We'll set it up to accept numbers with decimal points in them, convert them to long integers with a scale factor based on the number of places to the right of the decimal point, and then create a subroutine which can add any two scaled integers together without loss of precision. Finally, we'll set up a routine which uses "scale" and a "print using" spec in a string variable to print out the result with the correct number of decimal places.

First, the routine to input two numbers and do the conversion and scaling:

```
10   PRINT:INPUT"First number: ";a$
12   IF a$="" THEN END
15   GOSUB 905
17   IF errorcode THEN PRINT"Range exceed
     ed,try again.":GOTO 10
20   scale.first%=scale%:first&=a&
25   INPUT"Second number: ";a$
30   GOSUB 905
32   IF errorcode THEN PRINT"Range exceed
     ed,try again.":GOTO 25
35   scale.second%=scale%:second&=a&
40   PRINT USING 45;first&,scale.first%
45   IMAGE " first value= ",20#," scale
     factor= ",3#
50   PRINT USING 55;second&,scale.second%
55   IMAGE "second value= ",20#," scale
     factor= ",3#
60   END
899  REM
900  REM subroutine to convert input to
    long integer plus scale
905  errorcode=0:ON ERR errorcode=
     ERR:OFF ERR:RETURN
915  x=INSTR(a$,".")
920  IF x=0 THEN a&=CONV&(a$):scale
     %=0:OFF ERR:RETURN
925  scale%=-(LEN(a$)-x)
930  a$=MID$(a$,1,x-1)+MID$(a$,x+1)
935  a&=CONV&(a$):OFF ERR:RETURN
```

The subroutine is really pretty simple. It uses the trusty "instr" function in line 915 to look for a decimal point in the input string. If none is found (the number is an integer), then the string is converted to a long integer, the scale factor is set to zero, and a return is taken. Note that conversion errors (such as overflow) are handled by the "on err" statement, which passes back the errorcode to the calling program. If a decimal point is found, the scale factor is set to the number of digit positions from the point to the end of the string (line 925) and line 930 and 935 scrunch out the decimal point and convert the resulting integer to a long integer value. Once back in the input routine, the errorcode flag is checked, and if everything is okay, some simple "print using" statements print out the result for comparison. It should be noted that these routines are not bulletproof but were deliberately kept simple to illustrate the major points involved.

Now that we have long integer representations of these decimal numbers, with appropriate scale factors, it is possible to create a routine which will perform arithmetic on them, even though they may have different scale factors. The following routine will illustrate addition:

```
    1000  REM add a& and b& and return result
          in sum&
    1001  REM use scalea% and scaleb% to re-
          turn scalesum%
    1005  errorcode=0:ON ERR errorcode= ERR:
          OFF ERR:RETURN
    1010  IF scalea%=scaleb%  THEN  sum&
          =a&+b&:scalesum%=scalea%:OFF ERR:
          RETURN
    1020  IF scalea%>scaleb% THEN 1070
    1030  factor%=scaleb%-scalea%
    1040  b&=b&*CONV&(10©factor%)
    1050  sum&=a&+b&:scalesum%=scalea%:OFF
          ERR:RETURN
    1070  factor%=scalea%-scaleb%
    1080  a&=a&*CONV&(10©factor%)
    1090  sum&=a&+b&:scalesum%=scaleb%:OFF
          ERR:RETURN
```

The first thing checked for is if the two numbers have the same scale factor. If so, then simple addition is all that is required, and "scalesum%" (the resulting scale factor from the operation) is set to the common scale. If the scale factors are unequal, then the two scale factors must be adjusted to be the same by multiplying the one with the larger scale by the power of ten required to make them equal in scale. An example will clarify:

Initial no.	Integer value	Scale factor
12345.6789	123456789	-4
98765.43	9876543	-2

Obviously, just adding the two integers will produce meaningless results. But multiplying the second number by 100 and adjusting the scale factor correspondingly to -4 will make it possible to add them directly. The situation now looks like this:

New format	Integer value	Scale factor
12345.6789	123456789	-4
98765.4300	987654300	-4

The sum of the integer values is 1111111089 and, after applying the scale factor of -4, the result is 111111.1089. You should realize that most floating-point Basics, no matter how many digits they allow in double precision mode, have extreme difficulty with these types of problems. The reasons are complex, but they have to do with the fact that there are some decimal fractions which cannot be represented exactly with a binary floating-point (real) number. This leads to potential loss of precision in the last decimal place, rendering the answer inaccurate. While one place out of ten or fifteen might not be critical in an empirical scientific calculation, accountants are fussy about all the pennies (or in the example above, tenths of mils) adding up exactly. Note also that scale factors can just as easily be positive. That is, 567890000 could be represented as 56789 with a scale factor of 4. The principles of addition would work exactly the same as in the example with decimal fractions.

With the techniques described above, you can now figure out the way the subroutine works. One final note, though. In line 1040 and 1080 we use an expression "10©factor%" to represent the power of ten to be multiplied by the long integer value. Mixed mode expressions are not allowed between long integers and other data types, so the "conv&" function was used first to convert the power of ten expression to a long integer.

Now that we have a subroutine which will correctly add two scaled numbers, we can put it into our previous input program. The combination looks like this:

```
    5    PRINT"Test of extended precision add
          routines":PRINT
    10   PRINT:INPUT"First number: ";a$
    12    IF a$="" THEN END
    15    GOSUB 905
    17   IF errorcode THEN PRINT"Range exceed
```

```
       ed, try again.":GOTO 10
20     scale.first%=scale%:first&=a&
25     INPUT"Second number: ";a$
30     GOSUB 905
32    IF errorcode THEN PRINT"Range exceed
       ed, try again.":GOTO 25
35     scale.second%=scale%:second&=a&
40     PRINT USING 45;first&,scale.first%
45     IMAGE " first value= ",20#," scale
       factor=",3#
50     PRINT  USING  55;second&,scale.sec-
       ond%
55     IMAGE"second value=",20#,"scale fac
       tor= ",3#
60     scale%=scale.first%:scaleb%=scale.
       second%
65     a&=first&:b&=second&
70     GOSUB 1010
72     IF errorcode THEN PRINT"Range of
       precision exceeded, try again.":
       GOTO 10
75     PRINT"sum=  ";sum&;"  scale  factor
       =";scalesum%
105    GOTO 10
899    REM
900    REM subroutine to convert input to
       long integer plus scale
905    errorcode=0:ON  ERR  errorcode=
       ERR:OFF ERR:RETURN
915    x=INSTR(a$,".")
920    If x=0 THEN a&=CONV&(a$)
       :scale%=0:OFF ERR:RETURN
925    scale%=-(LEN(a$)-x)
930    a$=MID$(a$,1,x-1)+MID$(a$,x+1)
935    a&=CONV&(a$):OFF ERR:RETURN
999    REM
1000   REM add a& and b& and return re-
       sult in sum&
1001   REM use scale% and scaleb% to re-
       turn scalesum%
1005   errorcode=0:ON  ERR errorcode=
       ERR:OFF ERR:RETURN
1010   IF scalea%=scaleb% THEN sum&=
       a&+b&:scalesum%=scalea%:OFF
       ERR:RETURN
1020   IF scalea%>scaleb% THEN 1070
1030   factor%=scaleb%-scalea%
1040   b&=b&*CONV&(10©factor%)
1050   sum&=a&+b&:scalesum%=scale
       a%:OFF ERR:RETURN
1070   factor%=scalea%-scaleb%
1080   a&=a&*CONV&(10©factor%)
1090   sum&=a&+b&:scalesum
       %=scaleb%:OFF ERR:RETURN
```

Notice that, in addition to adding the subroutine at line 1000, we have some codes at 60 through 105 to set up the call to the subroutine and then print out the results. This is all fine, but this was supposed to be an exercise in advanced uses of the "print using" statement. An ideal use of "print using" here would be to print out the results of the addition, with the decimal point in the proper place. But, since our answers can range from nineteen digits to the left of the decimal place to nineteen digits to the right, and only a total of thirty-two positions are allowed in a single numeric "image" field, it is not possible to create a single format which will handle all possible variations. Here's where Business Basic's ability to have variable format definitions really comes in handy. The following routine can be added to the program above to print the result correctly, no matter what the scale factor:

```
80     x%=LEN(CONV$(sum&)):neg%=CONV
       %(sum&<0)
85     IF x%+scalesum%-neg%<=0 THEN
       form$="2#":ELSE:form$=CONV$(x%+
       scalesum%)+"#"
90     IF scalesum%>=0 THEN 97
95     form$=form$+"."+CONV$(ABS(scalesum
       %))+"#"
97     PRINT "scaled result of sum:";
100    PRINT USING form$; SCALE
       (scalesum%,sum&)
```

Line 80 gets the length of the number to be printed in "x%" and "neg%" is a flag to tell if the number is negative (the minus sign will require an extra position in the output). Line 85 uses this information, including the value of "scalesum%," to figure out how many positions are needed to the left of the decimal point. Line 85 then creates "form$," the output format specification, to match. Line 90 checks to see if "scalesum%" is positive (if value is a true integer). If so, it's finished. Otherwise, line 95 creates the rest of the format spec by including the proper number of positions to the right of the decimal point. Lines 97 and 100 then print out the long integer using the "scale" function to place the decimal point properly.

Voila! This routine should give exactly correct answers over its range of values. One thing you might want to add to help in tracing what the program is doing is to print out the value of "form$" along with the result in line 100. Also, for your personal entertainment, you might want to create subroutines for subtraction and multiplication. Division can be done using a combination of the "div" and "mod" operators, but you will become embroiled in what to do about rounding off the results of certain divisions. Multiplication has the virtue of being exactly correct within the possible range of values.

open apple gazette

1850 Union Street #494

San Francisco CA 94123

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 Street in San Francisco.

MEMBERSHIP

Annual membership dues are $30 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 endeavors to aid other Apple /// users through this educational publication - "OPEN APPLE GAZETTE." Address all inquiries to: Original Apple ///rs, 1850 Union Street #494, San Francisco, CA 94123.

REPRINT POLICY

All articles appearing in the Open Apple Gazette not copyrighted 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 copyrighted article may be obtained by writing to the author c/o the Original Apple ///rs.

ARTICLE SUBMISSION POLICY

The Open Apple Gazette welcomes any and all articles dealing with the Apple /// Computer and its associated hardware and software. Articles should be submitted on diskette as an ACSII text file, such as those produced by either Word Juggler, Apple Writer /// or the Pascal Editor. Typewritten double spaced articles are also acceptable.

OFFICERS

PRESIDENT	Don Norris	(415) 921-3774
VICE PRESIDENT	Kent Hockabout	(415)521-5414
TREASURER	Julia Amaral	(415) 383-3088
SECRETARY	Charles Coles	(415) 386-8623
CONSULTANTS	Ransom Fields	
	Ken Silverman	
ASSISTANT EDITOR	Stuart Henderson	

Back Issues

Volume 1, Number 1	$3.00 each
Volume 1, Numbers 2-6	$4.00 each

Mail Requests for Back Issues to:

Open Apple Gazette
1850 Union Street, #494
San Francisco, CA 94123

-///-