open apple gazette

Second Edition **Volume I, Number 2** **May/June 1982**

Speeding up your Apple ///

Recently while working with the Apple Writer Word Processing Language I ran across the command "PND". This command turns off video output. With video output off, the 6502B microprocessor runs approximately 20% faster. This means that WPL programs will execute faster.

The Standard Device Drivers Manual on page 139 says that by holding down the CONTROL key while typing 5 on the numeric keypad you can turn the video display on and off. So the WPL command "PND" is doing the same thing as CONTROL 5.

I use VisiCalc extensively in my life insurance business to do calculations for proposals. Some of my models are fairly large and require several minutes to recalculate. This raised the question would CONTROL 5 work with VisiCalc. After pouring through the VisiCalc manual I could find no reference to using CONTROL 5 to turn off video output to speed up recalculations.

The first time I used CONTROL 5 during a VisiCalc recalculation to turn off video output I thought I would have to manually turn video output back on using CONTROL 5 again. Well to my surprise the video output was restored automatically after the recalculation was complete.

About this same time I was starting to read the Apple /// Pascal manuals. The Introduction, Filer, and Editor manual explained on page 23 why the video output was automatically restored. It also explains why the recalculation runs faster. The explanations from page 23 the manual are as follows:

> Pressing CONTROL 5 causes output to the display screen to be suppressed. Pressing CONTROL 5 again resumes display updating. This function allows compilations, assemblies, and programs to run faster since no time is spent in updating the screen, and processor time can be used for other tasks.
>
> Note that any program that makes a read to the console will turn the screen back on, just like pressing CONTROL 5 the second time.

VisiCalc after completing the recalculation is looking for input from the keyboard and this explains why the video output is restored automatically.
Loading a model or data using DIF, into VisiCalc triggers a recalculation so using CONTROL 5 will speed up the recalculation during this process.

CONTROL 5 also speeds up the following prodedures:

- VisiCalc
  1. Moving columns or rows
  2. Inserting or deleting columns or rows
- Apple Writer
  1. Global search and replace
  2. Renumbering a mailing list
- Mail List Manager
  1. Sorts

So the next time you are working with that large VisiCalc model, Apple Writer text file, or mailing list use CONTROL 5 to speed things up.

by Don Norris

- /// -

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.

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.

OFFICERS

PRESIDENT	Don Norris	(415) 673-7635
VICE PRESIDENT	Kent Hockabout	(415) 521-1771
TREASURER	Julia Amaral	
SECRETARY	Charles Coles	(415) 386-8623
MEMBERSHIP CHAIRMAN	Dave Meyer	(415) 573-5556
CONSULTANTS	Randy Fields Ken Silverman	

- /// -

Twenty Four

This our second edition is twenty-four pages long. It is an indication of the information that is becoming available about the /// from /// users. Your user comments and suggestions will help the Open Apple Gazette become even bigger and of more value to you.

You will note there are a few blank spaces this issue. The printing costs are such that it is more economical to print 24 pages than 20 pages. Your comments and opinions will fill them up in future issues.

Your contributions, ideas and suggestions are needed to enable us to provide you with the kind of information and services you want.

- /// -

Pascal Apple /// Terminal Program

Reproduced with the permission of Stephen Lloyd as originally printed in the San Francisco Apple Core Cider Press.

Some time ago, I presented a simple terminal program for the Apple][. This month, I have written a modified version of that program for the Apple ///. It has been modified to increase its efficiency, speed and features.

This project has its beginnings several months ago when I contracted to design some system software for the Apple ///. Since the only things I knew about this model of the Apple were some rumors started by their chief competitors, I had quite a bit of self education to do. The preliminary results of this education are an ever increasing respect for the system architects at Apple and a knowledge that the detractors of the Apple /// have not taken the time to understand its capabilities.

I recently had a discussion about the Apple /// with a PhD friend of mine in San Luis Obispo, CA. Although he does not own one, he did express some negative views about the capabilities of the Sophisticated Operating System (SOS [pronounced 'sauce'] as it is referred to by Apple) and its ease of use. After a lengthy debate, it became obvious that his arguments were based on experience he had gained in the days of Apple's 'red' book and in the less enlightened times of DOS 3.2.

He had two basic complaints; 1) SOS is not compatible with any version of DOS, and 2) The internal routines and locations used by SOS are not accessible to the programmer. Since I'd had similar objections when I first started using the Apple][language system, I was able to reply with the following suggestions.

To understand why SOS is not compatible with DOS, we must first examine a little of the apparent history of the Apple and its various operating systems. First came the cassettes. They were simple and cheap. The Monitor, Integer and Applesoft BASIC were designed with cassettes in mind. The immediate level command interpreter in any of these three languages was an integral part of the language being used. There were also commands executable by a running program which could save and load data on tape. Next came the disk systems. If you had one of the first versions, you will remember that the BASIC you used with cassettes was the same one you used with the disk system. This was done by a command interpreter within DOS which executed commands meant for DOS (such as catalog and bload) and sent all others to the BASIC interpreter. The problem came when a BASIC program was executing. Since there were no explicit commands in either of the BASICs, DOS had no way of knowing what to do. The solution to this problem was to let the DOS interpreter intercept all printed output from the program. Anything which was preceded by a control-D was executed by DOS. From the viewpoint of advanced operating systems, these techniques are very awkward and kludgy.

Next came the language system. With its 16 sector disks, it boosted storage capacity by over 20 percent. One of its problems was that it could not be used to read DOS format disks which were still stuck at 13 sectors. The solution to this problem was simple, release a new version of DOS (3.3) which would use 16 sector disks. This made the sector formats compatible between DOS and the language system, but it still didn't allow the language system to directly use DOS disks.

Recently, there has been a flurry of programs which allow translation between DOS format files and language system format files. Depending on your inclination, you can either spend many hours of typing and correcting typos from one of the various articles, or you can spend anywhere from $50 to $150 to purchase the required utility. The result is the same, you are adding another patch to the DOS kludge.

The second objection my PhD friend had was probably due to his desire to know more about the operating system than was needed to accomplish the tasks at hand. I recall my first experience with DOS and binary files. After a binary file had been on the disk for more than a day, the two questions which always came up were: 1) How many bytes long is the file? and 2) Where does it go in memory? Both of these questions could be answered easily by PEEKing at certain memory locations. This technique is typical of the way information is stolen from DOS. There are very few 'regular' DOS interfaces to provide the required information. Most of them depend on fixed locations in memory.

Although this technique works very well, it does impose severe restrictions on the kinds of changes Apple can make to improve DOS. If any of these special locations are moved, lots of programs will wander off into never never land when they don't get the right information. If these locations remain constant in all future versions of DOS, valuable memory will be lost by having to jump around them. Clearly, a better method of communicating with the operating system is needed.

Before I go any further I should clarify what I mean by a 'better method'. The past has seen several different operating systems (OS) for the Apple. In addition to the ones mentioned above, Forth and CP/M come to mind. Each of these have

advantages and disadvantages, but mostly the latter.

An operating system should be similar to an orchestra conductor. It commands and controls all of the elements of the computer. It tells them when to become active and when to be silent. Communication between all of these elements and the user programs is also handled by the OS. It routes the data to and from devices such as printers, keyboards, disks, and modems. All of this should be accomplished automatically without intervention by the user's programs.

The less the user has to know about how and what the OS is doing, the better. Now comes the catch. If the user program needs to know what the OS is doing, the OS should be willing to give up this information without a fight. No peeks, no pokes, just ask. This type of interface is usually called a system call. If there is one feature which will make one OS better than another, it is an effective implementation of system calls.

SOS provides all of these features. It handles communications between the user programs and the elements of the Apple ///. It also provides the system calls necessary to control and determine the status of these elements. It even gives the user a uniform set of definitions for these system calls. Certainly, SOS is a better method.

To illustrate some of the features of SOS and the way it handles communication between a user program and the various devices attached to the Apple ///, I have written a terminal simulator in Pascal. It allows the Apple to communicate with any of the various timesharing systems available. I've been using it to communicate with an IBM 370 type system and Micronet.

The program uses the Apple ///'s built-in RS-232 interface for connection to an external modem. It also uses the built-in screen and keyboard along with the disk system for text file transfers. To allow SOS to use all of these devices, a file called SOS.DRIVER must be constructed using the SYSTEM UTILITIES diskette and the SYSTEM UTILITIES DATA diskette. The procedures required are detailed in chapter 2 of the Standard Device Drivers Manual. The specific drivers needed are .RS232, .CONSOLE, and one or more Disk III drives.

The main section of the program is repeated here for clarity of explanation.

```
begin Initialize;
repeat {until Finished}
        repeat {until Command}
                Process_Remote;
                Process_Local;
                until Command;
        Process_Command;
        until Quit or Exit;
Finalize;
end.
```

I have tried to structure this program as much as possible. Most of the procedures have a beginning, a middle, and an end. The main procedure is no exception. The beginning, Initialize, sets up all of the constants, opens the required files and defines which devices will be used. The end, Finalize, closes files and cleans up the buffers used by the RS-232 driver. The middle is where all of the work gets done.

In the middle, the repeat..until Quit or Exit loop continually processes characters from the keyboard and RS-232 interface and takes care of any keyboard command which may be detected. Characters which come in from the keyboard are sent to the RS-232 interface by the procedure Process_Local. Characters which come in from the RS-232 interface are sent to the screen by the procedure Process_Remote. These two procedures are repeated until a command is detected by Process_Local in which case Process_Command provides the necessary actions and then returns to the above loop.

Most of the I/O done by this program uses unit procedures instead of the familiar reads and writes. This is done to decrease the overhead required for I/O and increase the data rate of the terminal program. An additional unit procedure, unitstatus, is used for communication with the operating system. This is the method provided by Pascal which allows system calls to be made to SOS.

Operation

To use this program , be certain you have a modem connected to the RS-232 connector on the rear of the Apple ///. The modem can be virtually any type which will operate at 300 baud. I normally use an acousti-coupler, but a direct connect modem will work just as well.

Now eX)ecute 'terminal'. After a few moments, the screen will clear and the cursor will appear in the lower left hand corner of the screen. When this happens, dial the appropriate number for the computer you wish to use. After it answers, simply use the Apple ///'s keyboard just as you would a terminal.

All of the characters on the keyboard can be used for communication with the computer. The computer can send any of the 128 available in the ASCII character set. The Apple will only display those which have values greater than 31. The characters below 32 are called control characters and have special effects on the Apple. One of these is the 'return' character. It causes the cursor to return to the begining of the line. Another is the 'bell' character. It causes the Apple to beep. The exact control characters that this program uses are indicated

in the main part of the procedure Display.

In all cases but one, you simply have to press the key to have it transmitted. The exception is the 'escape' key. This is used to indicate to the program that the user is requesting a special function. To send an escape, you must press this key twice. Once to indicate that you are requesting a special function and again to indicate that the function is the transmission of the escape character.

The other functions which are available are shown in the main section of the procedure Process_Command. They are divided into three different groups.

1) text file transfers
2) protocol functions
3) stopping the program

Transmission of text files is accomplished by pressing 'escape' and then 't' (an escape-t sequence). The program then requests the name of the file to be sent. You must respond with the complete path name of the file, includiing the '.text' extension. After you press 'return', the file is located and transmission is begun. If you wish to prematurely terminate the transmission, simply press 'escape'.

When the transmission is complete or 'escape' is pressed, the keyboard is again able to send characters to the computer. During transmission of the file, the keyboard is locked (except the 'escape' key) and will not respond.

Reception of text files is accomplished by an escape-c sequence. The program will respond with the state of the screen copy procedure, either on or off. If the copy is on, then every character displayed on the screen will be written directly to the file specified in the Screen_Name variable.

While the screen copy is on, you will notice that the program seems to pause while the characters in the file buffer are being written on the disk. This is normal and does not cause the loss of any characters sent from the computer. The reason for this is that the RS-232 interface is interrupt driven and receives characters even when the Apple is busy doing other things.

The protocol used to communicate with the computer can be set by any of the following:

Full duplex	escape f
Half duplex	escape h
Simplex	escape s

The full duplex protocol is used by many time sharing systems. Micronet and Tymshare are good examples of this. It allows characters to be sent and received at the same time. The characters typed at the keyboard are not sent to the screen, instead they are sent directly to the computer which then has the responsibility of displaying them on the screen. Among other functions, this allows passwords and other things which should be kept secure to not be displayed on the screen.

The half duplex protocol is used by many of the IBM based time sharing systems. It requires that the characters typed at the keyboard be immediately displayed on the screen. It also requires that characters only be sent in one direction at a time.

To do this, the keyboard must be locked at the end of every line (when a return is pressed) and unlocked when the computer is ready to accept more input. To do this, the computer sends a unique unlocking character immediately before it is ready to accept input. In this program the variable Unlock_Character is used for unlocking the keyboard. While the keyboard is locked, it will not respond to any characters except the command sequences.

The Simplex protocol allows characters to be sent in both directions simultaneously. It is similar to the half duplex protocol, but in this case the characters typed at the keyboard are also sent to the screen. The distant computer is not required to provide these characters. This mode is useful for communication between two Apple ///s using this program.

Stopping the program is done simply by an escape q sequence. It closes and locks the file used for copying the screen. Then returns to the Pascal system level.

Conclusion

I have been using this program for the past two months. As bugs have shown up, they have been eliminated. Since most of the program is well structured, these modifications have been relatively easy. Future additions are just as easy to make.

In addition to serving well as a terminal program, it also has allowed me to transfer text from one computer to another. This is especially convenient because not all of the people I must deal with have access to the same computer.

```
program Terminal;

const   Screen_Width = 79;
        Screen_Length = 23;

var     Command,Quit,Exit : boolean;
        Cursor_Vertical,
        Cursor_Horizontal : integer;
        Cursor_On,Escape : char;
        Display_FF : (Advance,Home,Clear);
        Protocol
         : (Simplex,Full_Duplex,Half_Duplex);
        Keyboard : (Locked,Unlocked);
        Lock_Character,Unlock_Character : char;
```

```
        Screen_Name,Printer_Name : string;
        Screen_Log,Printer_Log : boolean;
        Screen_File : text;
        Remote_File : interactive;
        Remote_Name : string[128];
        Send_Text_File : boolean;
        Local_File : interactive;
        Local_Name : string[128];

procedure Clear_Screen;
forward;

procedure Display(var Character : char);
forward;

segment procedure Initialize;

var Character : char;
    Control : integer;

begin
Screen_Log := false; Screen_Name :=
'SCREEN.TEXT';
rewrite(Screen_File,Screen_Name);
Protocol := Half_Duplex;
Send_Text_File := false; Local_Name := '';
Display_FF := Clear; Cursor_Vertical := 0;
Cursor_Horizontal := 0;
Keyboard := Unlocked; Lock_Character := chr(13);

Unlock_Character := '>';
Escape := chr(27); Cursor_On := chr(5);
Command := false; Exit := false; Quit := false;
Clear_Screen; unitwrite(1,Cursor_On,1);
end;

procedure Erase_EOL;

var Output_Line : array[0..1] of char;

begin
Output_Line[0] := chr(31);
Output_Line[1] := Cursor_On;
unitwrite(1,Output_Line[0],2,8);
Cursor_Horizontal := 0;
end;

procedure Clear_Screen;

var Character : char;

begin
Character := chr(28);
unitwrite(1,Character,1);
unitwrite(1,Cursor_On,1);
Cursor_Vertical := 23; Cursor_Horizontal := 0;
gotoxy(Cursor_Horizontal,Cursor_Vertical);
end;

procedure Display;

        procedure Write_Character;

        begin
        if Cursor_Horizontal<Screen_Width
                then begin
                unitwrite(1,Character,1,8);
                unitwrite(1,Cursor_On,1,8);
                Cursor_Horizontal
                  := Cursor_Horizontal + 1;
                end;
        end;

        procedure Bell;

        begin
        unitwrite(1,Character,1,8);
        unitwrite(1,Cursor_On,1,8);
        end;

        procedure Backspace;

        var Output_Line :
              packed array[0..3] of char;

        begin
        if Cursor_Horizontal>0
            then begin
            Output_Line[0] := chr(8);
            Output_Line[2] := Output_Line[0];
            Output_Line[1] := ' ';
            Output_Line[3] := Cursor_On;
            unitwrite(1,Output_Line[0],4,8);
            Cursor_Horizontal
              := Cursor_Horizontal - 1;
            end;
        end;

        procedure Linefeed;

        begin
        unitwrite(1,Character,1,8);
        unitwrite(1,Cursor_On,1,8);
        if Cursor_Vertical<Screen_Length
                then Cursor_Vertical
                  := Cursor_Vertical + 1;
        end;

        procedure Return;

        var Output_Line : array[0..2] of char;

        begin
        Output_Line[0] := chr(24);
        Output_Line[1] := chr(0);
        Output_Line[2] := Cursor_On;
        unitwrite(1,Output_Line[0],3,8);
        Cursor_Horizontal := 0;
        end;

        procedure Form_Feed;

        begin
        case Display_FF of
          Home    : begin
                    unitwrite(1,Character,1,8);
                    unitwrite(1,Cursor_On,1,8);
                    Cursor_Vertical := 0;
                    Cursor_Horizontal := 0;
                    end;
          Clear   : Clear_Screen;
          Advance : Line_Feed;
          end;
        end;

begin
```

```
if Screen_Log then write(Screen_File,Character);
if (Character in [' '..'~'])
     and (Cursor_Horizontal<Screen_Width)
        then Write_Character
        else case ord(Character) of
                7  : Bell;
                8  : Backspace;
                10 : Linefeed;
                12 : Formfeed;
                13 : Return;
                end;
end;

procedure Send(Character : char);

begin
case Protocol of
        Simplex     : begin

unitwrite(8,Character,1,12);
                      Display(Character);
                      end;
        Full_Duplex :
unitwrite(8,Character,1,12);
        Half_Duplex : begin

unitwrite(8,Character,1,12);
                      Display(Character);
                      if
Character=Lock_Character
                        then Keyboard := Locked;
                      end;
        end;
end;

procedure Process_Remote;

type Status_Info = set of (Output_Buffer_Size,
                           Output_Characters,
                           Input_Buffer_Size,
                           Input_Characters);

var Status_List
      :
array[Output_Buffer_Size..Input_Characters]
         of integer;
    Remote_Character : char;

begin
unitstatus(7,Status_List,13);
if Status_List[Input_Characters]<>0
        then begin
        unitread(7,Remote_Character,1,12);
        Remote_Character
          := chr(ord(Remote_Character) mod 128);
        Display(Remote_Character);
        if (Remote_Character=Unlock_Character)
             and (Protocol=Half_Duplex)
                then Keyboard := Unlocked;
        end;
end;

procedure Process_Local;

var Keys_Available : integer;
    Character : char;

        procedure Get_File_Character;
        begin
        if (Keyboard=Unlocked)
             or (Protocol<>Half_Duplex)
          then begin
          if Send_Text_File
            then begin
            read(Local_File,Character);
            if eoln(Local_File)
              then begin
              Character := chr(13);
              if Protocol=Half_Duplex
                then Keyboard := Locked;
              end;
            if eof(Local_File)
              then Send_Text_File := false;
            Send(Character);
            end
            else if Keys_Available<>0
              then Send(Character);
          end;
        end;

begin
unitstatus(2,Keys_Available,21);
if Keys_Available<>0
  then begin
  unitread(2,Character,1,4);
  Character := chr(ord(Character) mod 128);
  if Character=Escape
    then if Send_Text_File
      then Send_Text_File := false
      else Command := true
    else Get_File_Character;
  end
  else Get_File_Character;
end;

procedure Process_Command;

var Character : char;

        procedure Break;

        var Character : char;

        begin
        unitstatus(8,Character,2);
         { flush output buffer }
        Character := chr(2);
        unitstatus(8,Character,14);
         { force communications break }
        end;

        procedure Transmit;

        begin
        gotoxy(0,0); Erase_EOL;
        gotoxy(0,1); Erase_EOL; gotoxy(0,0);
        write('File name : ');
Readln(Local_Name);
        close(Local_File);
        (*$I-*)
        reset(Local_File,Local_Name);
        (*$I+*)
        if ioresult<>0
          then begin
          gotoxy(0,0); EraseEOL;
```

```
          write(Local_Name,' does not exist.');
          end
          else Send_Text_File := true;

gotoxy(Cursor_Horizontal,Cursor_Vertical);
        end;

        procedure Set_Screen_Copy;

        begin
        gotoxy(0,0); Erase_EOL;
        gotoxy(0,1); Erase_EOL; gotoxy(0,0);
        Screen_Log := not Screen_Log;
        write('The screen is ');
        if Screen_Log then else write('not ');
        write('being copied.');

gotoxy(Cursor_Horizontal,Cursor_Vertical);
        end;

begin
Command := false;
unitread(2,Character,1,4);
case Character of
        'b','B' : Break;
                { force a break onto the
                  communication line fo 500
                  millisec }
        'c','C' : Set_Screen_Copy;
                { turns screen copy facility
                  on or off }
        'e','E' : Exit := true;
                { flush remote in and remote out
                  buffers immediately  terminate
                  program without waiting useful
                  for malfunctioning host
                  computers }
        'f','F' : Protocol := Full_Duplex;
                { set host protocol to full
                  duplex }
        'h','H' : Protocol := Half_Duplex;
                { set host protocol to half
                  duplex }
        'q','Q' : Quit := true;
                { wait for remote in and remote
                  out buffers to empty terminate
                  program normally }
        's','S' : Protocol := Simplex;
                { set host protocol to simplex }
        't','T' : Transmit;
                { send characters from a disk
file }
        'u','U' : Keyboard := Unlocked;
                { for half duplex protocol,
unlock a
                  locked keyboard useful when
                  Unlock_Character gets lost in
                  communications }
        end;
if Character=Escape then Send(Escape);
unitwrite(1,Cursor_On,1,8);
end;

procedure Finalize;

var Character : char;

begin
if Exit
        then begin
        Character := chr(0); {null effect}
        unitstatus(7,Character,2);
         { flush input buffer }
        unitstatus(8,Character,2);
         { flush output buffer }
        end;
close(Screen_File,lock);
unitstatus(1,Character,7);
end;

begin
Initialize;
repeat {until Finished}
        repeat {until Command}
                Process_Remote; Process_Local;
                until Command;
        Process_Command;
        until Quit or Exit;
Finalize;
end.
```

- /// -

The Third Basic

By: Taylor Pohlman
Reprinted from Softalk Magazine

Welcome to a series of articles on Apple /// Business Basic, to the powerful new cousin to Applesoft, the extended Basic that many of you know and love on the Apple II.

My goal in this series is to make Business Basic a useful, familiar tool for you. To do this, I'll pass along ideas that will help make the task of creating applications programs simpler and more efficient. Because Business Basic and the Apple /// itself are new to many of you, we'll relate programming hints and techniques for Business Basic to the more familiar environment of Applesoft. To get the most out of this series, you should be fairly familiar with Basic language commands and keywords and be able to create simple programs. Without those skills as a starting point, this series would quickly grow into the equivalent of a serialization of War and Peace.

If you are not that familiar with Basic, your best bet is to start with the Applesoft Tutorial manual. If you have an Apple ///, simply boot the emulation mode disk, select the applesoft option, and insert the DOS 3.3 Master Diskette. Presto, you are now in Applesoft and can follow the Tutorial's instructions to get up to speed in Basic. Once you are familiar with Basic and its syntax (a word you are guaranteed to encounter in learning the language), you'll be ready to rip through these articles.

If you are already familiar with Applesoft or another Basic, you should be ready to dig right in to Business Basic. The series will assume that you have an Apple /// in front of you to try out all the things we'll discuss. For those of you in that fortunate position, the fun is just starting.

Many of you have an Apple II and are wondering if you need a /// for that big new application or as an office complement to your Apple II at home. For you, this series should reveal the power of the Apple /// and its relationship to the II. Hopefully, that will help you make your decision. Others of you will just be wondering what all the fuss is about, and for you we wish happy reading. No matter what your situation, you should be able to gain an understanding of the power of Business Basic and pick up some hints you can use in programming.

In any case, we welcome your comments, suggestion, gripes, or whatever concerning this column and Business Basic in general. If you've written interesting routines you'd like to share, have converted programs from another variety of Basic, or simply would like to do a core dump about your favorite subject, write to Open Apple Gazette. Items of general interest will find their way into these pages, ensuring immortality for both of us.

One last comment should be made, especially to those who aren't business programmers. Why is Business Basic named Business Basic? As any product manager will tell you, dreaming up a product name ranks with dodging trolley cars and escaping from Alcatraz on the all-time "must do" list. Thus it was with Business Basic. Certainly it's true that scientists, engineers, educators, hobbyists, and lots more of you who are writing nonbusiness applications will find just what you need in Business Basic. As you stick with us in this and coming articles, however, you'll see that many of Business Basic's most powerful features were specifically designed to meet the needs of business applications and permit the easy conversion of programs written in other business-oriented Basic dialects.

One of the other things we'll do along our way is to show how syntax in some of these other Basics can be translated to Business Basic. This will help you use the many reference and tutorial manuals on the market that use examples from other versions of Basic. We'll include tips on converting from Basic dialects found on minicomputers and mainframes.

Well, so much for preparation. Now let's get a look at this dragon we're about to slay.

Setting the Stage. Like any other sophisticated computer system, the Apple /// takes a layered approach to the operating system, languages, and utilities that animate its hardware. The term layered refers to the several levels of software that insulate users from needing to know exact details of the hardware on which their programs are running.

Apple ///'s operating system is known as SOS (pronounced "sauce"), which stands for Sophisticated Operating System. The origin of the name is curious. Several years ago in the development of the Apple ///, the project was given the code name of Sara, named after the daughter of one of its inventors. Thus SOS originally stood for "Sara Operating System." When the time came to make it an official product, the name SOS stuck, so the marketing department had to come up with another word starting with S that made sense. That's how Apple ///'s operating system became "Sophisticated." As we explore more of SOS's capabilities, we hope you'll agree that it deserves the name.

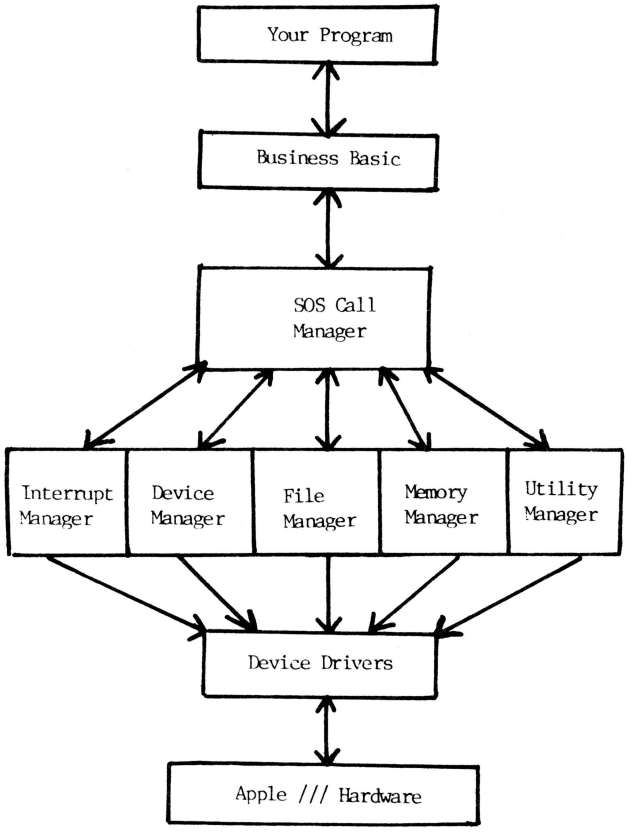
SOS's layered approach to system control makes it more than just a disk manager (like DOS) or an I/O convention (as are IN# and PR#). SOS truly manages all of the Apple /// system resources to simplify a programmer's life.

In Apple ///'s SOS, the lowest level of software is the hardware driver. The term driver may seem strange, but it's very logical. Just as the driver of a car has to know the operational details of what's being driven, so the Apple /// drivers need specific information about how the device is connected, what its features are, how it's controlled, and what information must pass back and forth between the device and the next highest level in the system. The beauty of this scheme is that the driver can be known by some generic name (like ".PRINTER" or ".TCLOCK") so that the operating system and Business Basic can use the device without being concerned about all the specific information that the driver must know. For example, you don't need to know anything about transmissions and turn signals to take a cab across Manhattan (a paid-up insurance policy will suffice). To extend the metaphor even further, you don't even have to know what taxi company to use; they all work pretty much the same.

In the same way, SOS can reference a ".PRINTER" for you, which may be a Centronics, a Silentype, and Epson, a Qume, or any of numerous other printer, connected via parallel, joystick, or serial ports. The higher you get in the operating system layer, the less specific you must be about the resources you use since SOS knows about all the devices you've configured on your system. Facilities are also provided to allow managing devices on a demand basis (that is, when they signal that they want to do something, called an interrupt). This feature makes it possible to request that more than one device be active at a time. To do that on the Apple II takes some pretty sophisticated programming.

Because of SOS's structured, layered nature, activities like reading from a remote computer while writing a message to disk and printing out a report become almost trivial.

We'll look at more about that later. It's sufficient for now that your program runs in Basic, which runs on SOS, which controls the hardware drivers, which accomplish the input/output to receive and deliver data for the system's devices (including a device called ".CONSOLE," which is the keyboard and screen). The structure looks something like this:



As you can see, each layer depends on the one below for services. Since the way the layers communicate is standardized on the Apple ///, it's possible to make substantial changes to the hardware and even to some parts of the operating system without changing the way Business Basic operates. This ensures that your programs will continue to work, even if we make changes later. Designing operating systems this way takes longer and makes them larger, but, in the long run, the benefits are enormous.

Getting Started. Since booting a disk is worth a thousand "you're gonna love its," let's get started by trying some things out. Just put the Business Basic disk in the built-in drive and press reset while holding down the control key (called "control-reset" from here on).

The first thing you may see is a slight flicker as the onboard diagnostics check out the Apple /// circuitry. Next is the SOS display screen, which indicates that the operating system has been loaded into memory. SOS's next task is to load the language from the boot disk. Since this is the Business Basic disk, that language is loaded and the hello program is automatically run (just like DOS in the Apple II).

You'll note that the final thing to appear is the right parenthesis")". This is the Basic prompt, meaning that Business Basic is ready for a command.

At this point, enjoy yourself for a minute by typing:

) PRINT FRE

You 128K Apple /// owners will notice that you've got more than 70K of user space for programs and data. We'll find some fun things to do with all that room later. The line shown also illustrates another convention we'll be using throughout these articles. What you type will always be underlined to distinguish your commands from what the computer outputs to you.

There are several items of interest in the display of the catalog. First, in the upper lefthand corner of the printout is the name Basic. This can vary from disk to disk and is called the volume name. SOS identifies the diskette you're referencing by a scheme called the Pathname. The highest level of the Pathname is the volume name, with any subdirectories mentioned next and the actual file name last (lowest) in the hierarchy. More on the subject appears in the Apple /// Owner's Guide and Business Basic manual under "Pathnames."

The next thing to notice is the column on file type. The type SYSTEM is obvious; that's SOS and Basic, the system software. Notice that Basic is named "SOS INTERP," because on this diskette, it is the interpreter (control

program) currently configured to run on SOS. Notice also that the "BLKS" column shows the space occupied on the disk in blocks. There are 512 bytes in each block. The next columns, alas, alack, are only relevant to those of you who have working clock chips. The files in the Business Basic disk directory will be marked with the date and time of their origin, but, without a system clock, the files you create will not. The final column, EOF, lists the exact number of bytes occupied by the file.

Now back to the TYPE column for minute. It's easy to figure out that file type BASIC stands for' a Basic program (like TIMESET). What does PASCOD stand for? Right, it's a Pascal code file, in this particular case created by the Pascal system's assembler. As you might have guessed (if you've been reading your Basic manual), the INV suffix on those files is a way of indicating that these files are set up as Basic Invokable Modules. We'll explore these in more detail later, but for now just remember that Basic uses assembly language routines through a mechanism called Invoke and Perform. There are some definite rules to follow in setting up these modules, which we won't go into now. However, there's no reason why we can't start using these capabilities right away! Hang on for a short exercise in using the SOS file system, and we'll give READCRT. INV a workout.

To get a glimpse of how Business Basic works with SOS to manage system resourses through files, let's take a simple example that doesn't require the disk or a printer. Basic tells SOS that it wants to use a file by means of the OPEN command and assigns a number for later reference to the file. On the Apple ///, of course, everything is treated as a file, even the keyboard and display. As we said earlier, the keyboard/display device is referred to as .CONSOLE. Note that the names for all character devices-devices that transmit one character at a time-start with a period. Type in the following so we can experiment (as Dr. Frankenstein said to Igor):

```
) 10 OPEN#1, ".console"   (This sets up a file number for Basic to use in communicating to the console.)
```

Note that you're already communicating to and from the console. That's because the console is the "default" I/O device. Statement 10 establishes a second path by which to communicate to the same device.

```
) 20 INPUT a$             (This is the good old ordinary input to the default input device.)
) 30 PRINT a$             (Again, default output device is the screen)
) 40 PRINT#1; a$          (Now we print to the screen again, this time through the console file previously opened.)
) 50 INPUT#1; a$          (This time we input from the keyboard, using the console file.)
) 60 PRINT a$             (Print to default screen.)
) 70 PRINT#1; a$          (Print the same quantity to the console file.)
) 80 END
```

Now if you LIST and RUN the result, it should look something like this:

```
)list
10  OPEN#1,".console"
20  INPUT a$
30  PRINT a$
40  PRINT#1;a$
50  INPUT#1;a$
60  PRINT a$
70  PRINT#1;a$
80  END

)run
?hello default console
hello default console
hello default console
hello  console  as  a filehello console as a
file
hello console as a file
```

A couple of interesting things are apparent here. First, although the first three lines work exactly as you would expect, the next three lines of output are a little different. The default console prints the question mark, as it should, but on line 4 of the output there is no question mark or prompt for input at all. This is because SOS is treating the console as a general input file and therefore can't know that it can accept characters printed to it. It just does a read to the device and waits for an end of record character (in this case a carriage return). The second unusual thing is also on line 4 - the PRINT command in statement 60 prints right at the end of the input string (unlike line 2). The same reason applies since the carriage return you typed and the subsequent line feed the system generates for the default console are suppressed for an input file device. But line 5 is printed separately, since the PRINT command in statement 60 outputs a carriage return and line feed.

In this same way, every device connected to the Apple /// is available as a file. The ability to address the console devices separately will come in handy in some future articles.

Having experimented a little with files, let's use one of those invokable modules we mentioned earlier and the OPEN statement to do something useful. This is a handy utility to use to make printouts of the screen when something strange or wonderful happens.

In this example, I'm assuming that your printer is a Silentype. Since SOS doesn't care what device it writes to, you may substitute any output file name in line 100, even a disk text file.

```
)new
)100 OPEN#1,".silentype"
)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
)1000 VPOS=23:HPOS=1
)1010 CLOSE
)1020 END
```

Listing this program should show:

```
100  Open#1,".silentype"
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
1000 VPOS=23:HPOS=1
1010 END
```

Notice that this reveals another nice feature of Business Basic: it automatically indents FOR-NEXT loops for clarity. Ever been jealous of those pretty Pascal listings? Business Basic to the rescue!

On a more serious note, let's look at what this program does. After OPENing the appropriate file in line 100, Basic is told to INVOKE the file readcrt. inv. Readcrt.inv is an assembly language' routine that looks at the current position of the cursor. The cursor position is defined by the current values of the Basic reserved variables HPOS and VPOS. Readcrt. inv then modifies the value of the variable "value%" to contain the decimal value of the ASCII character at that location. The INVOKE command tells Business Basic to find a place for readcrt.inv in memory and to set up a table of all its PERFORMable routines. You can INVOKE any number of modules, and Basic will always ensure that they are located in noninterfering areas of memory.

Line 120 sets up a loop that will scan the vertical lines of the screen. Line 140 sets up the inner loop which will look at every horizontal character position on that line. The routine in readcrt. inv is then called using the PERFORM command. Isn't this easier than a bunch of pokes and a call? Line 170 prints the character equivalent to file one, our output file, and then takes a look at the next position. Line 190 makes sure we print a carriage return at the end of each output line (since that character isn't physically on the screen). After that, line 200 starts scanning the next line. Lines 1000 through 1020 set the cursor at the screen bottom, close the output file, and end.

Now for the fun. Run this program and you'll get an exact copy of the first twenty-three lines of the screen on your output file. By putting in an INPUT statement to ask for the file name and then OPENing the resultant string variable as the file name in line 100, you can decide at the time you run where you want the copy to go. Use this program to document all the strange and wonderful things you find in Business Basic as you really begin to explore the language. But first, be sure to save the program to an initialized diskette!

Well, that's it for now. Until next time, happy coding with the most powerful BASIC around.

- /// -

FAITH HOPE and CHARITY

By: M. Kent Hockabout

Whether it was a problem looking for a solution or a solution looking for a problem, the rationale used by many of us who have purchased the Apple /// was not based on defensible logic. A good reputation, proven performance and a wealth of software are the sort of logical reasons one might assume to be a requirement for such a purchase.

It would have been nice to have such assurances, but they were hardly what we had to rely on for our decision. What were the reasons which made us do it? As much as anything, I believe it was an act of faith, a certain amount of hope and not an insignificant amount of charity.

Faith in the ability of Apple to replicate its Apple][successes. Hope in a favorable market for the /// against the coming tide of IBM, Xerox, NEC and all. And charity, in our willingness to forgive and forget the water under the bridge since the introduction of the ///.

So specifically why did I decide to buy the Apple /// ? The process began with the purchase of a TRS-80 Model I Level II 16K. After many volume settings between 4 and 6, I decided that there must be a better way of finding out how the computer could make work a little easier.

The alternatives seemed to boil down to machines of promise, compromise or the Apple][. The][seemed to offer the widest range of applications with an entry price which didn't look too bad. The price began to climb however, as I started tacking on the various features which seemed appealing, 10 key pad, 80 column card, modem, softcard and so on.

Without any prompting from the dealer - to my surprise - I suddenly realized that for a few more dollars I could buy a /// which had what I wanted and more. Although at the start I would never have thought of aiming at the ///, it now felt like the most natural thing to do. The machine looked good, felt good and the Visicalc program performed its miracles. Plus, I have always felt that the colorful Apple has a certain pizzaz not shared by either the Pet or TRS-80.

Thus, I became the possessor of an Apple ///, or the possessed if you listen to my wife, and the fun begins. I started reading all of the manuals at once. Visicalc was a snap to get up and running. Business Basic, on the other hand, only proved what was painfully obvious - the Lawrence Hall of Science beginning Basic course wasn't enough. What about graphics or is it graphix? Either way, graphics seemed to be a thing of the future. The success rate I had with the SYSTEM CONFIGURATION PROGRAM was not much better. And then La Machine proceeded to pop a chip.

In an attempt to reduce the mounting frustration, I began to reread the stack of magazines gathering complaints in the corner. Surely there would be a product review and perhaps an article or two from joyous owners which could solve some of the riddles. No, I did not go back to my dealer for help. The dealer had already sold me three programs which would not work in emulation so I figured we were in the same boat. (The dealer took back the programs.)

What little information I found was not very reassuring, reliability and supply problems. Apple Computer even dropped its ad which featured the ///. This was hardly a time of rejoicing about my decision. In spite of the picture in a weekly news magazine of what looked like thousands of Apple /// computers on an assembly line, I felt that I must be the only person to have bought an Apple ///.

Then there was the article in the International Apple Core magazine about the /// and the beginning of a users group, and the world didn't seem to be such a lonely place after all.

With the reintroduction of the Apple /// - PLUS SOFTWARE - and the beginning of the Apple ///rs, we're finally on our way. 'Course I always knew it was just a matter of time.

- /// -

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. 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

- /// -

APPLE /// PASCAL UTILITY LIBRARY

By William J. Cheeseman

After a long wait, my office finally received the Apple /// Pascal Utility Library. This is a brief note on its features, and instructions on converting it for use on the Apple][.

The package comes with two disks and a typewritten, spiral-bound manual. The manual is not a tutorial, but for the most part a procedure-by-procedure list of the input and output parameters of the utility procedures (and functions, of course). There are one or two brief examples of how to call most of the procedures and functions in your programs. The source text files for all of the utilities, as well as for several demonstration programs, are included on the disks. They are well commented, and thus provide a tutorial of sorts in their own right. A LIBRARY incorporating the utilities is included on one of the disks, so you can begin programming with them immediately, without any need to compile them or to construct the library yourself.

The utilities are organized into three separate UNITS. The first, GENUTIL, contains a hodge-podge of generally useful routines, ranging from the very complex to the ridiculously simple. It appears that several of the simplest routines (e.g., SOUNDBELL, which rings the bell once) are provided because they are needed by the more complex routines -- no quarrel with that, and if you would prefer them to work a little differently, all you have to do is edit the source code and recompile your own version.

This is not the place to list them all, but here are some examples which will give you the flavor:

1. PROMPT is an elaborate procedure for placing a prompt on the screen wherever you want it, defining default values, and formatting the desired response in any of 17 ways, including DOLLARS, STDDATE, etc. Special edit features using the OPENAPPLE key are provided for editing a response before it is finally sent to the computer, such as inserting characters at any point, jumping to the beginning or end of the response, restore to default, etc.

2. RESPOND is a simpler procedure to prompt for and act on yes/no and similar inputs.

3. GETCHR is a more elaborate version of everybody's getchr routine. Besides the usual testing against any defined set of allowed characters, it provides for optional delay and termination if a response is not received, bell, and so on. Optionally, the global variable ESCTYPED will be defined to aid in processing the ESCAPE key.

4. There are several standard screen and report formatting routines which I haven't yet explored, but which appear to be useful at least for programming quickies where you are willing to live with the built-in formats and conventions.

5. DATECOMPARE returns a -1, 0 or 1 to indicate whether a date is before, at or after another date. Too bad it doesn't tell you how far before or after.

6. VALIDDATE is a boolean function which tests a date string in the form MMDDYY for validity (Feb. 29 in a non-leap year, and so on). Several other procedures and functions allow input, output and compacted storage of dates.

7. FMTNUM is a procedure to convert a numeric string to any of eight provided formats, including DOLLAR, TRAILMINUS, COMMAS, and so on.

8. EVALINT and EVALUATE convert input strings to integers or long integers. Good tools for bullet proof input routines.

9. There are several other routines, including a number of string, character and numeric primitives to fill in the gaps in 'pure' Pascal.

The second UNIT is FILEACCESS, and the third is BTREE. I won't go into detail, as their purposes are apparent from their names. Suffice it to say that these routines appear to be more sophisticated and flexible than most. The file access routines handle most of the drudgery of opening, maintaining and closing files using virtually any record format you wish to design. Flexible file headers are provided for global file information, including an optional binary map of which records are active and which inactive. Files are sequential and untyped, for rapid storage and retrieval by block. The BTREE routines may be used for quickly locating and retrieving the record you want.

Several demonstration programs are provided in source and p-code, including a nice TESTUNIT which lets you try out each of the GENUTIL utilities interactively. There are 4 demonstration programs relating to file access, including a simple INVENTORY program.

I use my Apple][at home to write programs for my office Apple ///. Hence, I need to be able to use these Apple /// Pascal utilities on my Apple][. It turns out to be no problem (but a little tedious to set it up the first time!). The Apple /// library, when transferred bodily to an Apple][formatted disk, will not work correctly as is. You must recompile each of the source text files, either using the Apple][compiler option on your Apple///, or using your actual Apple][compiler after transferring the text files to an Apple][formatted disk. Then

construct your own Apple][SYSTEM LIBRARY in the usual way.

There is a kink, of course: the Apple /// Pascal screen control codes are not the same as those used by Apple][Pascal, but instead are the Apple /// SOS codes. Thus, you must make a couple of changes to the text files before you compile them for the Apple][. I use a Videx Videoterm 80-column card at home, which conforms to the Apple][Pascal specifications. You may have to do the following a little differently for your own Apple][setup. (I wish the program author had defined his screen and keyboard control codes in the global CONST declarations -- it makes modification for transportability much easier.)

1. In GENUTIL2.TEXT, the procedure INITCTRL must be changed so that CONST OPENAPPLE = -64. This enables the openapple edit keystrokes of procedure PROMPT to function with the equivalent CTRL characters; i.e., treat the Apple][CTRL key as if it were the Apple /// OPENAPPLE key.

2. In GENUTIL3.TEXT, the CASE options in procedure SCREENMSG must be changed to CLRLINE : WRITE(CHR(29)); and CLRSCREEN : WRITE(CHR(11));.

I am aware of no other required changes. The p-code files for the demonstration programs run on the Apple][without recompiling.

- /// -

User Comments On The Word Juggler

By: Dr. R. Smail ST0823

WORD JUGGLER is extremely user friendly. The program disks come with templates that fit over the Apple /// keyboard. The templates make using WJ as easy as playing an organ with lighted keys to show you what comes next. Printer format keys are on the top keyboard row and editing keys are on the numeric keypad. The layout is simple, powerful, and easy to learn.

All of the usual word processor commands are supported. These include block moves, loads, deletion-string search and replace etc. Text can be saved as an ASCII file for downloading over a modem. That should save some money when using data bases such as SOURCE. Text can be displayed in the form it will have at printout so that any changes can be made in column alignment etc. Printer enhancements such as multistrike and underline are supported. Text can be right or left justified, centered, or a semiproportional space. Automatic headers and page numbering are supported. Printout can be by entire document, one or several pages in a document, or multiple copies of any combination of these. Documents previously prepared can be inserted automatically at run time.

The software hooks are in place to support the internal clock if you are lucky enough to have one. A simple data file merge capability is provided on the WJ disk for the preparation of form letters and mailing labels. There is also a simple sort routine provided for the data files that are used in this manner.

I have used WJ to prepare over 100 pages of text so far. It seems to be most functional and is bug free as far as I can determine.

The spelling checker, LEXICHECK, is also excellent. Lexicheck is invoked with a single key stroke and works well with WJ. More about that later.

More specific detail on the WJ commands etc. can be obtained from QUARK.

I can answer any questions as a user of WJ (Smail ST0823), but I am no programmer and wouldn't be of much use regarding technical questions. I used AppleWriter /// at my dealer, and I liked WJ because it was user friendly.

I really only have two complaints about the total package.

First, WJ does not support true micro-spaced proportional printing (as yet) on my Qume Sprint 9. They use a semiproportional spacing by inserting extra spaces between words when "justify" is selected. This is a kind of micro-spaced emmulation....ok, but I'd like more. They say the programming for that is VERY

complicated and who am I to say different.

Second, footnote formatting and automatic insertion with text line count adjustment and automatic numbering (anything else) is not supported.

Now, for some good news. When I wrote to QUARK about these possible enhancements they replied that a new version of WORD JUGGLER is in the works which will include both true proportional spacing and automatic footnoting. If they can pull it off that will just about top it out for me with this word processor.

That is it for this file. Smail any remarks to {ST0823}.

I would be quite interested in incorporating useful user experience with WJ in updates to this file.

- /// -

MX-100 Manual Revisited

By: M. Kent Hockabout

In the Premier issue, all twelve type styles for the MX-100 were identified, as well as the necessary control/escape codes. However, the procedures for using these character types with both Apple Writer /// and Visicalc /// was not spelled out.

A number of calls have been received, one from Hawaii, asking about the specifics of how these codes are entered for both Apple Writer /// and Visicalc ///. So for those who have not found the right sections in the manuals or who have just bought a MX-100, here is how it is done.

----- APPLE WRITER /// -----

For those of you who like to follow along, turn to page 50 in your Apple Writer /// manual, top of the page starting with "CONTROL-V CONTROL-characters as Text Entries:". The CONTOL-V is toggled on and off in order to enter the Control/Escape codes. When you enter a CONTROL-V, a "V" appears in the Data Line just to the right of the "Z". Apple Writer /// allows the use of commands to the printer to be embedded in the text. If you wish to use the same character type for the whole document, place the Control/Escape code at the beginning of the text. I suggest that you get in the habit of turning off what ever character type you are using at the end of the text. The first article included the necessary codes. This is especially important if you are using more than one character type in a document.

Control Escape codes are entered as follows for Apple Writer ///:

<control>-V'control/escape-letter'
<control>-V

You will note, if everything is right, that the above procedure will place your print command for an <escape>-CAPITAL letter on the screen as an inverse left bracket "[" and a non-inverse capital letter. For a '<control>-letter' you will see an inverse capital letter. NOTE, if you make an error while entering these commands after the CONTROL-V, just enter another CONTROL-V and perform your usual deletion command to erase the error. If you attempt to correct an error with the CONTROL-V on, you will merely enter a string of control characters.

Once you start using the various combinations of character types in a document, you may notice strange things happening. Enlarged characters will throw off margins or centering and some character types just do not seen to work. The reasons for all of these strange goings-on will have to wait for another day. There are also the various embedded commands for form feeds and other printer commands in addition to these commands for type styles. If anyone feels like tackling the combination of formatting and type style commands, please do.

----- Visiclac /// -----

Now for Visicalc /// and its special commands for the printer. Turn to page 208 in your Visicalc /// manual, beginning with the heading "The Setup String." The sequence is as follows:

/PP"<return>.

This will produce the blank prompt line, ready for you to enter the desired Control/Escape codes. But before you enter the now-familiar codes, it is necessary to enter a carat "^". The "^" is a <shift>-6. With the "^" entered, you are ready to enter the Control/Escape codes.

As an aside, you may wish to mark your calendar so a year from now you may recall this occasion because we are in the "Good Ole Days" when each program had a different command structure.

The Control/Escape code for Emphasized is:

^<shift> E <shift> E <return>

The letters for each type style as well as the "C" for control and "E" for escape are capitalized. If you make an error or wish to change your mind, you DO NOT have to go back to the beginning. Enter the edit mode by entering control E, the using <escape> or control H to delete the unwanted characters. When you press <return> the prompt line will then request the lower right coordinate.

For those character types which are compound Control/Escape codes, you may enter each one individually, or all on the same line. If you wish to do everything at once, which is easier, seperate each Control/Escape code with a carat "^".

As mentioned before, the enlarged character automatically turns off at the end of the line. If you forget this piece of trivia, and you expect to have your complete spread sheet printed in condensed-enlarge (#7 in previous article) you may wonder what's wrong now.

However, this feature may be useful for printing a one-line-at-a-time title on a Visicalc report. Otherwise you would have to print the title line(s) first and then reset the character type to standard, if necessary, and then enter the character type you wish to use for the body of the model.

Experimentation is necessary if the type style used in the title is a different size than the type style used in the body.

In the "things move fast" department I mentioned the lack of italics for the MX-100 in the previous article. Within hours of the ink drying the EPSON ads appeared for the

GRAFTRAX-Plus giving Italics and other goodies to the MX-100.

- /// -

Manual & Documentation

User Comments from Sam H. Bell

Why Apple would produce a nice machine like the /// and not supply the user an owner-oriented publication to use it effectively and easily is beyond me. I refer to simple, complete, and useful examples of the machine's hardware and software functions. Their Business Basic reference manual is not a tutorial (Apple even says so) and one is sorely needed.

I am certain some group of Apple employees have the smarts and inside design details on these units but management or marketing does not make them available to the purchasing customer. If the "Open Apple Gazette" can change this, all of us will be better off.

Witness the head bashing that Mr. Norris spoke about in the March 1982 issue. We ran into this same problem with Business Basic and it is only Mr. Taylor Pohlmans recent series of Business Basic articles in "Softalk" magazine that have shed some light on this programming language nearly a year after the unit was introduced.

APPLE UPGRADABILITY

As a side issue, if Apple does not make the /// upgradable to Lisa or something close to it, I will be disappointed. I may have to buy one of their 68000 units anyway, but I will complain before I do so to their Cupertino headquarters.

DISK STORAGE

I have found that the 5 1/4 inch floppy disks are not of large enough capacity for serious business use. I sort of knew that before we purchased our /// but still hoped my instincts were wrong. The only simple alternative is a hard disc. Two brands known to me are off the shelf as of May 1982. Are there more?

1. I have looked into the Profile disk that Apple sells and cannot rationalize anything of 5 megabite capacity without realistic backup media built into it. I fault Apple for the engineering execution behind that piece of hardware.

2. For this reason, backup, the only disk I can justify is a Corvus with the Mirror video tape option. We have not bought one yet but if we do I will review its performance for the readers. I have heard only good about the Corvus reliability. Has anyone used it with their ///?

- /// -

APPLEWRITER ///

By: Dr. R. Smail STO823

APPLEWRITER /// is a vast improvement over the word processors (then really only text editors) found on small computers.

Commands are entered by control characters, which is fairly common (the other common method is the use of modes--in which you cannot insert unless you are in the insert mode, etc.). Deleting unwanted text is done with a CTRL-back arrow, which is perfectly logical, but does not allow for forward deletions. Retrieving lost (deleted) characters is done with CTRL-right arrow.

Word wrap-around is handled fairly well, with two glaring peculiarities. First, if a sentence, which is customarily started with two spaces, starts a line, it is preceded by a blank. Fortunately, it corrects this flaw when being printed. Second, when a segment of the text buffer is filled, all the screen will compact (at least this is the only explanation I can come up with for this bizzare occurance), and the cursor and text will jump--usually half a line. The text isn't harmed, it's merely visually distracting.

Most formatting functions are supported, and are implemented with menu commands or text-imbedded dot commands (commands that start a line with a period). Text can be formatted to print however you please. Some consider this a disadvantage, but I personally prefer this approach. A nice feature is that the printing defaults can be saved to files, so several different configurations can be ready-at-hand.

The load and save commands both have defaults. If an "=" is entered, when a file name is displayed on the Data Line, it means use the file listed on the Data Line. This procedure is not as nice as the scrolling of file names in VISICALC ///.

APPLEWRITER /// can also be used in conjunction with MAIL LIST MANAGER, but there is a conversion process required. The mail list cannot (to my knowledge, at least) be filtered in any way prior to conversion without creating another MLM diskette. Any field or combination of fields can be used anywhere in the APPLEWRITER file, which is a handy feature. I do not know if other word processing systems have this--the only other one I've seen only allowed the whole mailing label to be inserted.

Other unusual features: the fonts can be changed (four are provided), the screen can be split, a glossary of commonly-used terms can be created, control characters are shown on the screen when inserted into text, and it has its own word processing language (WPL) for batch jobs. There is also a provision for changing the case

(upper or lower) of any text.

All in all, Applewriter /// isn't perfect, but for $225 it is one of the better bargains I have seen. I have not experienced any flaws I couldn't live with.

- /// -

Business Basic Software Library

By: Stan Guidero

If you've had your Apple /// for awhile, you probably noticed that there is very little public domain software available. Well we hope to change that in the near future with your help.

What kind of programs do we need? Well, any program that you wrote (authored) in Business BASIC, Pascal or even COBOL, (the source code must be included with Pascal and COBOL programs), invokable modules, Visicalc templates or device drivers. The programs can be very simple or complicated. Games, utilities, demos, graphics, business or home programs are all acceptable and in demand.

If you wrote a small (or big) program to handle that special task, send it in. It might be just what someone was looking for. Incidently, there are many many programs written for other computers that may be easily changed to work on the Apple ///.

And now some suggestions or conventions, if you will on what we need in or with each program. DOCUMENTATION, documentation, and more documentation, inside and outside your program. A program may be useless if we don't know how to use it. A set of instructions within your program is very helpful. Your name should be included in the REM statements or in the title page.

Note: Please do not send in commercial copyright programs as we cannot use them.

To send in your program use a floppy disk mailer with a return address so we may return a disk with programs from our library. Your original program may not be returned. Please send to:

The Original Apple ///rs
Attention: Basic Librarian
P.O. Box 813
San Francisco, CA 94101

Then watch this space for the announcement of our first library disk which should sell for about $7.50. Between us we can develop the finest library of programs of any club.

- /// -

Letter From Apple ///r Ralph Merrikin

Dear Don:

Here is the $25 for membership and the comments on Apple I agreed to send. I enjoyed chatting with you at the Applefest 82 in Boston. I have the March issue of the Open Apple Gazette and if there are any since that time I would appreciate the back issues.

I have now owned four Apple ///s and feel that Apple has done me an injustice. The first three units I owned had many problems. This is not earth shaking news to anyone I'm sure! Like many others, I too spent many hours trying to get my system up and operating. The trips to the dealer were many, quite distant, and aggravating. This includes trips to the regional Apple office in Westboro Massachusetts to cure intermittant problems which the dealer was unable to correct.

And still I don't complain! But what really upsets me is after all my trials and good faith, Apple lowers the price of the system without compensation to the original owners. Just think....all I had to do was wait for a year while all you Apple-nuts debugged the product for Apple Company, and I could have saved wear and tear on my car, not used all that gasoline, not paid turnpike tolls, etc. And I would have gotten a working system at a much lower price!

Better yet, I would have gotten an All Apple system including the nice monitor which goes with the system. And my extended warranty would then cover the monitor. Are you early Apple /// owners aware that your Sanyo Monitor sold with the early systems is not covered by the Apple extended warranty, even though it came in a box with an Apple label?

Now I am willing to admit that the Apple /// is a very fine piece of engineering, and it will probably do everything I ever want to do.....but, how come the dealer keeps getting all the technical bulletins and I can't even get a return phone call from the "Hot- Line" I read about in the Apple advertisements? The dealer has the information I require, but because nobody at the dealership keeps up to date on the Apple /// the information is filed away and forgotten.

To all this I must add that I took my case directly to Apple via letters to all major officers of the corporation. I never got a single letter in return, nor phone call nor note...nothing! Almost a year of writing to Apple about the /// problems and never a response. I came to call this phenomenon the "Cupertino Black Hole." Everything goes in and nothing comes out!

At Applefest 82 in Boston, I grabbed a front

seat in the auditorium and waited for Steve Jobs to appear. At the proper Q & A time I stood and told Mr. Jobs that I had a suggestion to alleviate some grievances of the earlier Apple /// users. Mr. Jobs asked me to speak with him later and proceeded to rehash the same story about the early Apple /// manufacturing errors.

My suggestion to Mr. Jobs would have been to let the early Apple /// owners select software equal to the price difference between the new Apple /// and the early Apple ///. I will finish by adding that I waited 45 minutes for Mr. Jobs to finish giving autographs....at which time he walked away saying "I'll have your dealer get in touch with you about your problem." Would you buy a new computer model from Apple? Not Me!

Yours truly,
Ralph Merrikin
280 East Street
Brockton, MA 02402

- /// -

Changing The Text Mode On The Apple ///

By Paul S. Trueblood

The Apple /// comes with a standard display of 80 columns of black and white text, which is preferred in most applications. However, there are times when it is desirable to have larger characters on the display, to make it easier to read, or to have the characters appear in color to add interesting effects to programs, or to match the appearance of a graphics program. In such cases it is quite easy to tell the Apple /// to change the mode in which it is displaying text.

In a BASIC program, you must PRINT a control character to the console driver (chr$(16)), followed by a one character argument specifying in which mode you wish the text to be displayed. To change to 40 column black & white send chr$(0), to change to 40 column color send chr$(1). The following program demonstrates how to change text modes in a BASIC program:

```
)10  REM Clear textscreen
)20  Print chr$(28)

)30  REM Set text mode to 40x24 black & white
)40  Print chr$(16); chr$(0)
)45  Vpos=12
)50  Print"Now is the time for all great Apples"
)60  Get ch$
)65  REM clear textscreen
)70  Print chr$(28)
)90  REM Set text mode to 40x24 Color
)100  Print chr$(16);chr$(1)
)110  REM Set foreground color to yellow
)120  Print chr$(19);chr$(13)
)130  REM Set Background color to brown
)140  Print chr$(20);chr$(8)
)145  REM clear screen
)150  Print chr$(28)
)160  Vpos=12
)170  Print "When in the course of Apple events"
)180  get ch$
)185  REM set text mode to 80x24 black & white
)190  Print chr$(16);chr$(2)
)200  end
```

Changing the text mode from a PASCAL program requires a slightly different technique because the PASCAL language intercepts many of the control characters before they can get to the console driver. In order to smuggle these control characters through the Pascal interpreter they must be embedded inside a data structure, then transmitted to the console driver using the UNITWRITE procedure. The following PASCAL program illustrates the use of this technique:

```
Program Text_Mode_Demo;
var Smuggler:packed array[1..2] of 0..255;
begin
write (chr(28)); (clear text screen)

Smuggler [1]:=16; (informs console driver you
wish to change text modes)
Smuggler [2]:0; (selects 40x24 black & white)
UnitWrite(1,Smuggler,2,,12);   (sends   control
chars to console driver)
gotoxy(0,12);
writeln('Now is the time for all great Apples');
readln;
write(chr(28)); (clear text screen)
Smuggler[2]:=1; (selects 40x24 color)
UnitWrite (1,Smuggler,2,,12);
Smuggler[1]:=19;  (inform console driver you wish
to set foreground color)
Smuggler[2]:=13;  (selects  yellow  as foreground
color)
UnitWrite(1,Smuggler,2,,12);
Smuggler[1]:=20;  (inform console driver you wish
to set background color)
Smuggler[2]:=8;   (select   brown   as   background
color)
unitwrite(1,Smuggler,2,,12);
write(chr(28)); (clear text screen)
gotoxy(0,14);
writeln('When in the course of Apple events');
readln;
end.
```

If you run either of these programs you will notice that any text present on the screen when you change to a different text mode will revert to a strange combination of symbols and letters. It is therefore advisable that you clear the text mode options refer to the Apple /// Standard Device Drivers Manual, pages 34-42. The PASCAL UnitWrite procedure is discussed in the Pascal Programmer's Manual Volume 1, pages 207-210.

- /// -

Access ///

By: Dr. R. Smail ST0823

Access /// is a terminal software program supplied by Apple Computer for use with the Apple ///. It comes in the "Special Delivery" package - one diskette and a manual. The disk is not copy-locked.

This program can be used either from Business Basic or from Pascal. The diskette must be configured to your system and is not bootable as supplied. The user prepares a bootable diskette using the Apple supplied utilities programs. The process is not difficult and is clearly described.

System requirements include an Apple /// and a modem. Acoustic or direct connect modems (such as the Hayes SmartModem) are fine.

Access /// allows preparation and storage of data or text off-line. This allows down-loading during connect time at speeds of 110 to 9600 bits per second (software selectable) for considerable savings in usage fees on-line. This feature really makes one wish for a 1200 baud modem. Special protocols are not required as this is handled by Access ///.

Minimum memory required is 128K. The modem connects to the RS232 port - or, direct connection via this port from Apple /// using Access /// to a remote computer is also allowed. One problem with the use of the single RS232 port on the Apple /// for the modem is that a printer requiring this port can't be used at the same time for printing on-line. There are hardware solutions for that problem however. (ed. note: an Apple Universal Parallel Interface Card and the parallel version of the printer will get around this problem.)

Access /// allows recording of received transmissions into disk files when the user is on-line. The default recording file is on the internal disk and storage space there is limited when this is the boot diskette. Recording can be done on formatted storage disks prepared in advance. Formatting cannot be done from within Access ///. The recording file can be filtered (software toggle) to eliminate all received control characters with values less than ASCII 32 except for tab, return, and line feed.

Set-up mode allows selection of a wide range of options that allow specific configuration of ANSI mode, VT52 mode, LF after CR, 7 or 8 bits per character, enable/disable XON/XOFF, normal or inverse video, full or half duplex, normal or application keypad (allowing keypad sequences identical to those used by the VT100), normal or application cursor keys, wraparound if more than 80 characters are written per line, standard or graphics character set (replaces lower case with special symbols), speed, parity, and tab stops .

The current set-up configuration can be saved for automatic insertion on boot-up.

In transmission mode previously prepared files can be transmitted to the remote computer. Options include setting line delays of 0 to 25 milliseconds, and character delays of up to 255 milliseconds. These delays may be required by the remote computer (Source does not require them at 300 baud - I don't know about 1200 baud). The ESC key will abort transmission and return the user to terminal mode. On return to terminal mode the CRT screen will be blank except for the cursor - so, you must remember what you were going to do next after you send the data of text.

Files prepared using WORD JUGGLER or APPLEWRITER /// can be used as the transmission file without any difficulty.

There are some special considerations in using this program that have emerged so far. These are items known by this author. Other special problems may also exist and will turn up as user experience with this program grows.

USING ACCESS /// WITH SMARTMODEM:

It is very important to be sure the internal switch (S1) is set to pull DTR to logic TRUE. If you don't do this, or use a hardware fix that has been suggested, you CANNOT leave terminal mode without disconnecting. What happens here is that Access /// closes the RS232 driver and DTR goes FALSE. The Smartmodem then disconnects. The solution is simple : set S1 correctly and you are home free.

TAB SETTINGS:

Apparently you must set the tabs or remote tabs will scroll to the extreme right edge of the screen and the cursor will stay there until a CR is received. Again, the solution is simple : set your tabs. A tab set every 10 characters works fine for me.

Any user input re the use of this program for the Apple /// would be welcome. Smail to ST0823.

One last thought - I should have mentioned that the keyboard con-figuration can be changed. This downloadable feature allows use of the Dvorak keyboard for those of you who are progressive enough to be willing to relearn your typing.

- /// -

Word Juggler ///

User Comments from Sam H. Bell

Word Juggler /// is a word processing package which we like and found affordable. It was made available several months before the Apple Writer /// software last year (1981) and has been reliable since it was first up and running. We use it in a business application with all areas of customer service mailings, governmental forms, handings, and correspondence.

I would rate the Quark customer support excellent if our own experience is any measure of the service they give others. We have written them with several questions, and all have been answered quickly and correctly by return mail. I even called their office and was given the needed answers to my questions by their staff. Mr. Tim Gill of their office has gone out of his way to help us and when we needed a version of the program re-configured to support the Silentype printer, they did so at no cost to us.

The first time user of Word Juggler will find that the main menu has 9 features.

1. New : begin text entry mode
2. Catalog : of a disk
3. Load : file from a disk
4. Store : file to a disk
5. Purge : a file from a disk
6. Format : disk
7. Define Disk Drive Prefix
8. Edit Printer Configuration
9. Reboot

The program is copy-protected and therefore not easy to customize.

There can only be one choice of a printer on a single program disk at one time. Since we have to use the Silentype when our IDS 440 is in the shop, we have the Silentype drivers on our backup disk.

The Word Juggler program has combinations of express keys which give the user very fast movement through text. Whereas the IBM programs (especially the one that is currently the most popular) take forever to move up and down, the Word Juggler travels as fast as I can compose. No one should have to wait for a program to the point of boredom.

Another point that instantly endeared the Juggler to me was its solid error handling and recovery. I have never had the program bomb or hang in my 8 months of using it daily. Every mistake is recoverable with the escape key.

Quark's policy is to provide a backup disk at the time of purchase. This is a must for programs operating in a business environment such as ours where downtime must be avoided.

The operator's manual supplied with the disks is in the form of a loose leaf binder, and seems to be free of errors and typos. The instructions and tutorial are in conversational form. At times I would rather have step by step directions but again both my staff and I learned to use the program on the first try.

The best feature about the Juggler is one that Apple Writer/// lacks: a set of keyboard overlays. I cannot say enough about their usefulness to me. If you do not have such an aid, you must refer costantly to a manual. Quark has implemented both the escape/numerical keys above the keyboard and the shift/numerical keys on the keypad to the right side of the keyboard. This is good creative use of the ///'s capabilities, and about 40 special abilities are thus supported.

Most of the needed word processing functions one looks for are provided for in the Juggler package. Juggler runs with keys or inserted commands nested between and amongst the text being edited. Because of this what you see is not what you get unless you move to the display document mode. One then steps through the CRT display of the text a page at a time. On our 128 K machine we can edit a document of 799 lines of text in memory at once.

Functions provided for are:

- Centering
- Justification
- Allowing spaces at the end or beginning of a page
- Changing pitch for printing control
- Insertion of documents
- Replacing strings in text
- New Page form command
- Indenting
- Double spacing or single spacing
- Pause in output
- Setting page length, margins, width of text.
- Skipping lines
- Block stores and loads
- Find commands
- Display cursor movement key definitions
- Deletion of next and previous characters
- Deletion of word or line
- Print of Document
- Display of document
- Menu return

The new revision we have (2.2) supports more functions and special abilities which I will review when I am more familiar with them. If you need a word processor that you can depend on and that has good support, call or write to Quark for more details. They have a winner!

Sam Bell
1378 Freeport Rd.
Pittsburgh, PA. 15238

- /// -

WHY A ///

by Don Norris

A few weeks ago I attended IAC President Ken Silverman's housewarming in Santa Clara and visited with IAC Chairman Bernie Urban. When I told Bernie that the /// was my first computer and yes I had one of the earlier ///s which was subsequently replaced by Apple, he asked me would I buy a /// again. The answer is an emphatic yes.

Why? There were several reasons for my purchase in February 1981.

1. It was an Apple Computer, not a TR--S- ##, purchased from R-d-o, Shack. After all have you ever heard of the San Francisco TR(What Ever)## Core, or is there a Publication called the R-d-o Shack Orchard.

2. VisiCalc. As a life insurance agent working with lots of numbers I was looking for number crunching and VisiCalc on the /// was head and shoulders above VisiCalc on the][. Primarily because it was 80 full columns. Plus it had Upper and Lower Case. I ordered my /// from the Computer Connection here in San Francisco, and since my order was placed during some of the "turbulent" times of the ///, delivery of my /// was delayed. While I was waiting I used their demo /// (while it was healthy) or one of their][s in the store. Changing the direction of the cursor on the][with the space bar was a real pain.

3. Built in numeric keypad. I work with a lot of numbers and this was a must.

4. A word processor was supposed to be released shortly. (What ever happened to Word Painter anyway.)

5. In addition to the larger memory the /// is faster and easier to use than the][.

Perhaps some of my initial decision was based upon blind faith that it was not an Edsel, and would be supported by Apple. Additionally I believed other software producers would begin to produce software written to take full advantage of the features of the ///. The replacement of Apple ///'s with a serial number lower than 14000 tells me that Apple is commited to the ///. Software is being written by several companies now.

Admittedly my /// may not create the image of an integrated professional system with my Sanyo Monitor as it would with the Monitor ///, and my second disk drive is noisier and not as aesthectically pleasing as the new ones. The system works and has been a profitable addition to my business.

Owning an Apple and becoming involved with user groups I have greatly expanded my realm. I have made friends and contacts which would have been impossible otherwise. Additionally the intellectual challenge to learn to use even some of the computer's potential is the greatest one I have found.

- /// -

A CALENDAR FOR APPLE ///

Do you need a calendar for your Apple ///? This program written in Business Basic will print a calendar for you starting with the year and month of your choice. Known as APPLE CAL it was originally written for the][by Glen Teman and has been revised by Dwight Norris. The Apple][version was published in NIBBLE MAGAZINE Vol.3 No.2.
To get the program, send an address label, $3.00, and a disk with your best Apple /// program(s) on it to:

Dwight Norris
5295 Belle Isle Dr.
Dayton, Ohio
45439

The disk will be returned to you in a new library case. (As long as you are sending diskettes send one to our Business Basic Librarian. Ed.)

- /// -

Source Users Group

Several of the articles for this issue are used with permission from a users group on the Source. There comments are greatly appreciated. We were unable to obtain their names so we used their Source numbers.

- /// -

DIF /// to][

DB Master is a very popular Data Base program used by many Apple /// owners in the emulation mode. DB Master supports the Data Interchange Format, which means you can transfer information between it and VisiCalc.

Fine you say, but I want to use VisiCalc /// with all of its memory, etc.

The solution is to transfer the files from DOS to SOS using the text file converter since DIF is a text file. Those of you who own Apple Writer can use the Apple Writer utilities diskette to transfer files from DOS to SOS and from SOS to DOS.

- /// -

open apple gazette

P O BOX 813 SAN FRANCISCO 94101