6809 EXPRESS

The Official Publication of The

PENN-JERSEY COLOR COMPUTER CLUB

MAVERICK BBS 215-760-0456

JULY - AUGUST 1992

TREASURES ON TAPE

RICK HENGEVELD

Ahh, The dog days of summer are upon us! This is the time of year we want to just kick back and enjoy. No time for working.

Well we have a work project coming up at this months PJCCC meeting. The club library has had in its possession a large quantity of tape based programs. Since virtually nobody in the club uses tapes, we've decided to convert these tapes to disk. If you think you have a very good collection of Coco software and you've might already have these programs on disk, well guess again.

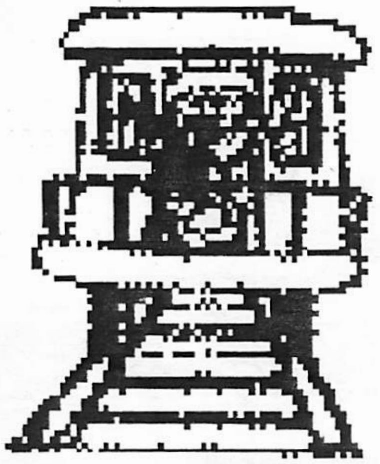
Most of these programs are titles that are rarely seen. There's also a good collection of old "Hot Coco on tape" in this collection! Many adventure games and some good work software.

So it looks like July will be transfer month. Also this will be a good opportunity to see how tape to disk transfers are done. Hope to see you there in our new meeting room, 190 in the main college hall.

****

THERE WILL BE NO ISSUE OF
THE 6809 EXPRESS
IN AUGUST, SO IF YOU GET
ONE, SOMETHING'S WRONG!
...Editor

****



THE LIBRARY CAR

ALAN J. WAGNER, SR.

Welcome to the July session of the PJCCC 6809 Express Library Car. First I would like to express my appollogies for the way the program printed. I had intended for the copyright statement to come out a little more organized. Starting this month I will be sending any Library Reports that contain programs to the editor in Max 10 format so that I can be a little more sure that the program will be a bit more readable. This time we are going to discuss the program to create the database and the program to delete an entire database. In addition, we will cover a little extra program needed to ensure that the directories we create will be in all capital letters. This is to comply with the non-binding rule of making all directories all capitals to make them easy to spot when you do a DIR command.

Let's start with the Makebase program. Last session we created a small dummy file called Makebase to test the Main Menu program. This time its for real.

```
PROCEDURE makebase
0000     REM ***********************
001A     REM *                     *
0034     REM *      BASIC09        *
004E     REM *                     *
0069     REM *     Make  Base      *
0083     REM *                     *
009D     REM *Copyright Jun 1992   *
00B7     REM *                     *
00D1     REM * Alan J. Wagner, Sr. *
00EB     REM *  222 Jefferson Ct.  *
0105     REM *  Quakertown, PA     *
011F     REM *        and          *
0139     REM *                     *
0153     REM *  The PennJersey     *
016D     REM *                     *
0187     REM *   Color Computer    *
01A1     REM *                     *
01BB     REM *        Club         *
01EF     REM **********************
0209     REM
020C     DIM current,filename:
          STRING[32]
021C     DIM x:REAL
0223     DIM oops,path:BYTE
022E     DIM ans:STRING[1]
023A      PRINT CHR$(12)
023F     RUN printat(28,5)
024A      PRINT "Home Inventory"
025C     RUN printat(28,7)
```

```
0267        PRINT "Create Database"
027A        RUN printat(26,9)
0285        PRINT "Copyright June 1992"
029C        RUN printat(23,11)
02A7         PRINT "Enter path for new
                                Database:"
02C7        RUN printat(28,13)
02D2        INPUT filename
02D7        IF filename="" THEN END
02E4        ENDIF
02E6        RUN makeupper(filename)
02F0        PRINT CHR$(12)
02F5        RUN printat(20,5)
0300         PRINT "Checking to see if ";
                                filename; "
exists."
0326        ON ERROR GOTO 10
032C        FOR x=1 TO 4000
033F        NEXT x
034A        OPEN #path,filename:
                                READ+DIR
0356        RUN printat(20,7)
0361          PRINT filename; " already
                                exists"
0378        RUN printat(20,8)
0383          PRINT "Returning to Main
                                Menu"
039D        CLOSE #path
03A3        FOR x=1 TO 8000
03B6        NEXT x
03C1        END
03C3 10     oops:=ERR
03CC        IF oops<>216 THEN
03D8          RUN printat(20,11)
03E3              PRINT "Error number ";

                                     oops; "
has occured."
0408          RUN printat(20,12)
0413              PRINT "Check out what


happened, then try
                                            again."
043F              RUN printat(20,14)
044A                  PRINT "Press ENTER to

                                              return to
Main Menu.";
0472              INPUT ans
0477              END
0479            ENDIF
047B            PRINT CHR$(12)
0480            RUN printat(23,5)
048B                  PRINT "Creating New

                                          Database
Directory"
04AE            RUN printat(28,7)
04B9            PRINT filename
04BE            SHELL "makdir "+filename
04CD                  CREATE #path,filename+


"/Dbase":WRITE
04E2            CLOSE #path
04E8                  CREATE #path,filename+


"/mfr.idx":WRITE
04FF            CLOSE #path
0505                  CREATE #path,filename+


"/dscrip.idx":WRITE
051F            CLOSE #path
0525                  CREATE #path,filename+


"/wherepurc.idx":WRITE
0542            CLOSE #path
0548                  CREATE #path,filename+


"/location.idx":WRITE
0564            CLOSE #path
```

```
056A      PRINT CHR$(12)
056F      RUN printat(25,5)
057A      PRINT filename; " directory
                             and     files
created."
059F      FOR x=1 TO 6000
05B2      NEXT x
05BD      END
```

Much of what I've done this time has been covered before. Note the line that begins at offset 02D7. Here we check to see if the filename given is longer than a single carriage return. If it is, we end the program and return to the Main Menu. Two lines further down we run a program called "makeupper" and pass it the variable filename.

```
PROCEDURE makeupper
0000        PARAM target:STRING
0007        DIM x:INTEGER
000E        FOR x=1 TO LEN(target)
0020        IF ASC(MID$(target,x,1))
              <=122 AND ASC(MID$
              (target,x,1))>=97 THEN
0041        IF x=1 THEN
004D            target:=CHR$(ASC(MID$
                (target,x,1))-32)+
                RIGHT$(target,
                LEN(target) -x)
006D        ELSE
0071            target:=LEFT$(target,
                x-1)+CHR$(ASC(MID$
                (target,x,1))-32)+
                RIGHT$(target,LEN
                (target)-x)
009C            ENDIF
009E          ENDIF
00A0        NEXT x
00A7        END
```

Note how when we declare the variable target to bea param and a string, we don't declare the length. Basic09 automatically declares the string storage space to be 32 bytes long. Don't confuse the length of the storage space with the length of the string. If you look at the line that starts at offset 000E, we have a statement LEN(target). This returns the length of string target. If the length of the storage were the same as the length of the string, this would always return the number 32, but it doesn't. What it returns is the actual length of the data in the variable target. For instance if we had declared the filename to be CLYDE, the statement LEN(target) would return the number 5. This then gives us a way to look at each byte of the string without wasting effort looking through blank or garbage areas of the varible storage.
In the next line we check to see where in the ASCII code that byte is located. The statement ASC(string) returns the ASCII value of the first character in the string. We will be interested in more than the first character of target, so I've used the command MID$(string,start,quantity) to return to ASC a one character string obtained x characters into the string varible target.
122 is the ASCII value for the letter "z". 97 is the ASCII value for the letter "a". If the value returned by ASC is equal to either or between these values, then we have to act on it because it is a letter, but is

not a capital. If this is the first letter, it is a special case as there are no letters to its left. Using LEFT$ and RIGHT$, we disect target and reassemble it with the letter in question converted to a capital.

This admittedly gets a little complex, but stick with me and I think I can walk you through it. Since we have several statements or commands nested inside each other, it is best to analyze them from the inside out. This is the way the computer will execute them. In the middle is the same statement we had a line or two earlier. The MID$(target,x,1) returns a single character string that the ASC, that precedes it, converts to a number equal to the string's ASCII value. On the other end of this parenthetical statement we find "-32". The numerical difference between a small letter and its capital equivalent happens to be decimal 32. By subtracting 32 from the ASCII value of a small letter we arrive at its capital ASCII value. The CHR$ now converts this converted value back to a string character. The plus sign concatenates this character with the rest of the target variable. RIGHT$(stirng,number) returns the "number" of characters from the righthand end of the string. Since we want only those characters to the right of wherever the xth character is, we can subtract x from the length of the string and it will return just what we need.

If this is not the first character, then the program proceeds to the line at offset 0071. This line is the same as the last one discussed except for the LEFT$ statement. Once having progressed beyond the first character, we have to account for the characters to the left of the xth charcter we are now examining. LEFT$ works simialr to RIGHT$ except that it returns the number of characters from the lefthand end of the named string. By subtracting one from x, we get all the characters to the left of the xth character. By concatenating the LEFT$, the MID$ that we converted, and the RIGHT$, we have converted the xth character and reassembled the string target back to its original form but now partially capitalized. This process continues until the end of the string is found by the FOR/NEXT loop.

An interesting experiment is to add a line "PRINT target" just before the "NEXT x" statement. Each time through the loop the statement will print the current target string. You will see the letters capitalized before your very eyes, but don't blink. Basic09 is very fast! You can run this experiment by booting Basic09, making the experimental change to the program, then typing 'run makeupper("a string")'. "A string" must be in quotes, but can be any string up to 32 characters long. The single quotes are just to set the command line apart from the text in this document and

should NOT be typed on the command line in Basic09.
Now that we've taken makeupper apart, let's get back to makebase. On the line that starts at an offset of 0326, we encounter yet another new command. ON ERROR GOTO allows us to trap system errors that would otherwise crash the program. If an error occurs, the program immediately goes to the line called out in this command. Then we can put a routine there to handle the error.

Next we find a little FOR/NEXT timing loop. I found that the following sequence of events can happen so fast that you don't get to read the statement that we are checking for the files existance.
Next we OPEN a path to read a directory filename. To open a path requires a variable in which to store the path number returned by the command. This path number is used to refer to the file opened. Several paths can be opened at once and can even be to the same file if needed. Back in the DIM statements we dimensioned a variable called path as a byte. Since the number of paths are limited to a maximum less than 256, we will be quite safe with this variable type.
The filename has been defined several lines earlier and would be the name of the file you wish to open.
After the colon, there are several options Basic09 allows. You can read, write, update, exec, and/or dir. Read and write mean just what you might think. If you anticipate having to do both with a file, open the file for update. If the file is in the current execution directory, open the file with exec. If the file you are opening is a directory, use dir. Since we don't want to take a chance at damaging any data, opening the file with a read option is a safe way to go. Also if we find the file, we expect it to be a directory, so the dir option is appropriate.

We really want this to fail with an error 216, "File not found" because we are attempting to create a new file. Just in case we do find it, the next couple of lines report that condition, CLOSE the path, give us time to read the message, and return us to the Main Menu. Notice the close command uses the path variable to close the specific path we just opened.
As I said before, we really are expecting an error 216. So, starting with line 10 (the only numbered line in this whole program), we attempt to handle any errors that may try to trip us up. In line 10, we equate oops to ERR. ERR is the inate variable in Basic09 that captures the last error number. ERR can be accessed only once before Basic09 resets it. To be sure we don't loose the number before we are done with it, it is always a good practice to equate it to a less volatile variable.
Since we expect error 216, we have to account for all the others in an intelligent way. If its not 216, the

program reports it, tells us to check it out, and then returns us to the Main Menu when we are ready. This allows the person running the program time to think about what happened, record the error number for future reference and then return to the main menu.

If it was an error 216, then the if statement of the last several lines falls through and the program clears the screen and tells us it is creating the new database directory.

The command SHELL, is the way Basic09 accesses the OS9 shell in which it is itself running. To use shell, you enter shell and then a string that would be the command you would type if you were at the OS9 command prompt outside of Basic09. In our case here, the command is MAKDIR. But makdir requires a name or path for the new directory. Our variable filename is already a string, so we concatenate the variable to the quoted string and the whole thing gets sent to the shell.

Now that we have our newly created directory, we have to populate it with the files we will need to operate our database. The create command works exatly like the open command, with the exception that it is used exclusively to create a new file. All of the options mentioned for open will work with create, although creating a file in the read mode seems a little silly since you can't read a still empty file.

Once the program has created all the files, it announces that fact and gives you time to read the announcement.

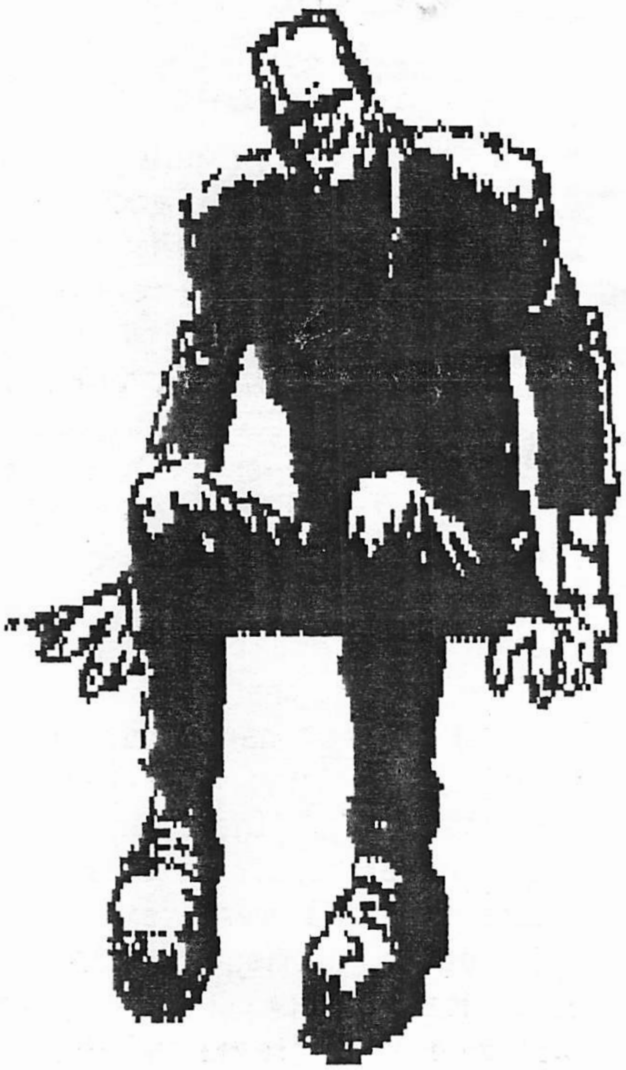
Now we have the problem of what to do with a database we no longer want around. The following is basedelete.

```
PROCEDURE basedelete
0000      DIM filename:STRING[21]
000C      DIM errnum,path:BYTE
0017      DIM ans:STRING[1]
0023      PRINT CHR$(12)
0028      RUN printat(25,5)
0033      PRINT "Delete database"
0046      RUN printat(25,7)
0051      PRINT "Enter filename to be
          deleted "
0072      RUN printat(25,8)
007D      INPUT filename
0082      IF filename="" THEN
008E        END
0090      ENDIF
0092      ON ERROR GOTO 10
0098      OPEN #path,filename:DIR
00A4      CLOSE #path
00AA      SHELL "deldir "+filename
00B9      END
00BB 10   RUN printat(25,10)
00C9      PRINT CHR$(3)
00CE      RUN printat(25,10)
00D9      errnum:=ERR
00DF      IF errnum=216 THEN
00EB        PRINT "File not found!"
00FE        PRINT CHR$(3)
0103        RUN printat(25,11)
010E        PRINT "Returning to Main
            Menu."
0129        FOR x=1 TO 8000
013C        NEXT x
0147        END
0149      ELSE
```

```
014D      IF errnum=1 THEN
0159        PRINT CHR$(12)
015E        RUN printat(25,11)
0169            PRINT "Returning to
                                 Main
Menu."
0184        FOR x=1 TO 8000
0197        NEXT x
01A2        END
01A4       ENDIF
01A6     ENDIF
01A8     PRINT CHR$(12)
01AD     RUN printat(25,10)
01B8         PRINT "Error number ";
                          errnum; "
occured."
01D9     RUN printat(25,11)
01E4      PRINT "Check out the error
                          and    try
again."
020A     RUN printat(25,14)
0215          INPUT "Press enter to
                          return   to
Main Menu.",ans
0240     END
```

There is really nothing new in this program. If you examine it, you'll find that many of the routines used here have appeared in the previous programs. At offset 00AA, we've used another OS9 shell command. DELDIR has its own "are you sure" inate to it. As such I felt we didn't need yet another one written into our program.

That wraps it up for this session. In the next session we will cover the update and view section and that will pretty much complete this program. I would like very much to hear from those of you who are reading this series and know if you are interested enough that you would like a disk containing all of the various procedures that go in to make up this program. I will make my files available to any PJCCC member who wishes to make a copy. Anyone else can aquire a copy by contacting the editor and the PJCCC will provide a copy for cost plus a small handling charge that will go towards the club's treasury.



OUR ASSISTANT TREASURER

The Maverick Report

RICK HENGEVELD

The Maverick is closing in on 700 calls since the system went up! The Maverick continues to perform it's main tasks of assisting in the publication of the **6809 Express** and linking fellow Coco users together.

Seems **Richard Kravitz** has gotten the hang of tele-comm as I keep seeing his name on the logs, good work Richard! And our Newsletter Editor, **Pete Unks** has checked in under MM-1 power! I guess persistence pays off.

Speaking of persistence I've managed to finally gather all the hardware to install a hard drive system on the Maverick. Now after I manage to talk a certain resident OS9 expert into configuring some software we may soon get our act completely together and expand the Maverick to what I envisioned it could be - a full service BBS.

Basically Speaking

RICK HENGEVELD

Good News, bad news time. First the bad news (For yours truly) the Coco cash register/database program I spent a month writing won't be needed by my video company. The good news is that I will not let that work go to waste. I intend to use the program as a tutorial of RSDOS basic and share it with 6809 readers. I hope you agree that this is good news!

We'll get to the actual coding next month, but first we'll explore some reasons you might want to dig a little deeper into basic than you may have already gone. By far, most people run commercial programs written by the Pros, Many times these software packages are written in machine code and perform flawlessly. So why should you pound the books and keyboard to create your own?

Well many times you have needs that no programmer could anticipate without full knowledge of your needs. In the case of the cash register/database program that we will look at, there were no commercial software packages available that could do exactly what I needed. Therefore it became time to roll my own. Fortunately a database program is a fairly simple task to write in basic.

My personal method of writing software is to break the process down into several steps. This way what may look like a huge task suddenly become very manageable. So here are the steps I usually follow.

Step 1. Define your needs, Sounds simple but this step is often the toughest hurdle to get past. Trying to foresee all the
possible needs and functions of a computer program is a tricky business. Even the professional programmer has difficulty with this, hence we see version 1.0, 1.2, 2.0 of the same program. Versions 1.2 and 2.0 are usually created after a programmer was heard to say "Nuts, I never thought of that"

Step 2. Decide the methods you will use to achieve your goals. Often there are options to consider, in the case of a database you may create files with either a direct or a sequential access system. Speed and simplicity may be considerations. Will you be the only person using the program? If so you can make the program fairly complex for the end user. If other people, particularly non-computer people will be using the program then you may need the program to be "Idiot-proof" Planning ahead for these things will make writing the actual program a simpler task.

Step 3. Once Steps 1 and 2 are complete I break down the tasks I need the computer to do into blocks or modules. In the case of a Cash Register/Data base program I broke the program down to 4 modules.

1. Cash Register, this section would handle the pricing calculations.

2. A database to keep logs of a customer name and address along with their phone number and the product purchased.

3. Printing routines to print both mailing labels and master lists of our customers along with master reports foe total sale figures.

4. Setup and screen display. This section would handle screen width and color along with printer pokes and screen menus.

By breaking the program down like this, no section ended up being more than 15 or 20 lines long. This keeps anything from getting out of control.

Well the word counter says I've given you enough to chew on for awhile, Next month we'll start to put together the program one section at a time along with an explanation of how it works. In the mean time dig out your Coco manuals and blow off some of the dust! You just might find you get a lot of pride in rolling your own, and it's easy!

A PREVIEW OF COMING ATTRACTIONS !

The Editorial Staff is Delighted to Announce These Features in the September Issue of the 6809 EXPRESS !

The Final Chapter of
Al Wagner's Basic09
DATABASE PROGRAM !

and (as if that weren't enough)

Peter Unks' Review of
the OS9-68000 Computer
THE MM/1

plus

DATAWINDOWS FOR OSK

TasCom FOR OSK

and

PRESTO PARTNER

Miss these at the peril of your future Happiness!

FIRST CLASS MAIL

H. Peter Unks, Editor

6809 EXPRESS

The Official Publication of The

PENN-JERSEY COLOR COMPUTER CLUB

Eric Rhyder
[redacted]
[redacted]

[illegible]

WILKES-BARRE PA 187

LEHIGH VALLEY PA
JUL 28
1992
180

6809 EXPRESS

The Official Publication of The

PENN-JERSEY COLOR COMPUTER CLUB

6809 EXPRESS

The "6809 EXPRESS" is the official publication of the PENN-JERSEY COLOR COMPUTER CLUB. The club is based in the greater Lehigh Valley of Northeastern Pennsylvania including sections of Northwestern New Jersey. Any non-profit organization may reprint any part of this newsletter provided credit is given. PJCCC will gladly exchange newsletters with any other computer club. Send requests to EDITOR, 6809 EXPRESS, Penn-Jersey Color Computer Club, 145 Seventh Street, Phillipsburg, NJ 08865. PJCCC assumes no responsibility for errors or omissions. PJCCC assumes no liability for damages resulting from the use of any information or programs contained in this newsletter.

GET ONLINE WITH THE MAVERICK
BBS! DIAL 215-760-0456