the MONITOR

December, 1992 — Commodore Users Group of Saskatchewan — Vol. 7, No. 10

Obligatory Stuff

Editorial

by Jarrett Currie

I would like to take the opportunity to welcome the new Executive to their new positions. Of course, I would especially like to thank Garth Strawford for letting his name be nominated for the position of editor. I am sure he will continue to enhance the Monitor so that it continues to be a vital part of CUGS.

I think that over the last three years that I have been editor, the most common theme in my editorials has been requests for articles. This month, only Barry, our club president, managed to write something for the Monitor. The other two articles were downloaded from CompuServe. For those of you who download programs from various sources, please remember that if you find a text file on a BBS about something of interest to Commodore computer uses, that it is fair game as a submission to the Monitor. Throughout my stint as editor, I have used articles from CompuServe, QLink and other Commodore clubs' newsletters, so I hope that as I stand down, that others will continue to disseminate information they find outside of Regina.

As this is my last editorial, I would like to thank those who made the position easier and more enjoyable. These are the people who wrote articles for the newsletter, and submitted them on time. I would also like to wish Garth the best of luck. I hope he finds the experience as pleasurable as I did.

The Write Disk

by Barry Bircher

Last month, I talked a bit about error handling routines and how to talk to the drive. This month I would like to discuss uses of the disk status byte "ST". The status variable has some very unique uses as it relates to accessing the disk drive. The disk status byte is read just like any other variable. The variable ST is special to the system and is a reserved variable. This means that it is a read-only variable; you can read it as a variable, but you cannot write to it.

When programming, it is not always known how big a file is when you read it in for the first time. A good example are seq text files from word processors. How would you know when the last character was read from the file? If you don't check for the last byte, you could be reading some extraneous information that means nothing (basically pulling data out of the air).

This is where the reserved system variable ST comes in handy. ST stands for disk STatus (serial I/O actually, which includes the tape drive). By checking ST each time you retrieve data from a file, you can tell when the last byte has been retrieved. The variable ST is set to 0 (zero) by the system in the background when things are OK and changes to 64 when the last byte of data was retrieved.

When you issue a BASIC command like PRINT ST, the system actually PEEKs a memory location (hex $90) which stores an 8-bit byte resulting from the last serial (tape or disk in/out) operation performed by the system. Each bit of the 8-bit byte means something. We are primarily concerned with the 6th bit. If the 6th bit is set, this gives the ST variable a value of 64 meaning that the end of the file has been reached. If you look at this variable right after a disk access, this variable will reflect the status of the operation. Generally speaking, ST will be 0 if the last access went OK. It will be something other than 0 if there was an error. If you keep checking ST when reading from a file, it will change magically (the operating system changes it) to 64 when the last byte was read. There are other numbers it will change to, but these are for more advanced uses and will be discussed later on.

The usual programming practice is to simply check to see if ST equals 0 and stop when it's not. Since our previously discussed program uses a GOSUB 1000 to check for errors, this is as good a place as any to put it, so we'll place it there as the first thing to check. We'll copy our version of ST and save it to the S variable.

The placement of the "S=ST" is important. We need to save our version of ST before any other serial operation takes place, including the error reading routine that is about to take place. ST will reflect how the last serial operation went. If the reading of the error channel is OK, then ST will reflect the true operation of the last serial operation and not our previous operation. So we just make a copy of it before the operating system decides to change it on us.

The following is a program I have written that I use as an example in this series of articles. Please note that this is a renumbered version with some slight modifications from previous articles.

```
10 rem open error channel
20 open15,8,15
30 rem open a file up
40 open 2,8,2,"unknown file,s,r":
50 rem check for disk errors like "FILE NOT FOUND"
60 gosub1000:if ds then 140
70 rem get a character from the file and check for errors.
80 get#2,a$:gosub 1000
90 if ds>0 then print"Disk error, program aborted":goto140
100 print a$;
110 if s=0 then 80
120 rem if program control has reached to here, then the file
has ended normally
130 print"No more info, file ended!"
140 close2:close15
150 end
999 rem disk error checking routine
1000 s=st:input#15,ds,ds$,tr,sc
1010 if ds <>0 then print ds;ds$;tr;sc
1020 return
```

In line 80 we retrieve a byte from the file in A$, then call

The President's Message

Hello, and welcome new members and executives alike. It has been a really short month since the last meeting and another one is upon us again. This month's presentation will see Tristan Miller doing a presentation on the CMD Hard Drive and showing how easy it is to add to your system. Also he plans on showing the speed of the drive in a graphic demo. I know this will be one meeting you will be sorry to miss.

There are several megabytes of Commodore files floating around the city and CUGS is going to get them for you. Stan and Tristan have volunteered to get some of these programs and put them into the library so CUGS members can get them as well. We are still working on the Florida disks (13 megs), however the going is slow because most of them are copyrighted, so we are wading through them and pulling out the public domain and shareware programs. This is a long and time consuming process of copying, unarcing, delynxing, extracting and checking out if the program first of all works, and to see if it is PD or shareware. I understand that the FACT BBS has just acquired well over 150 megs of files just waiting for Commodore users to use. These files should eventually filter down into our own BBS and library. I do see several hundred disks of new software coming out to the CUGS library in the coming years(s).

Well, till next time, have a good one.

on a subroutine (GOSUB) 1000 that does our disk error checking for us. It saves our copy of ST in variable S and checks the error channel. After the subroutine has "RETURNED" it checks if there was an error. If there was an error, then it prints out a message to the user that we are going to do something about this error. In this case, I chose to simply abort the program and close some files we had opened up and end the program (GOTO 140). If there was no error, then line 100 is executed and prints the character contained in A$ out to the screen. Line 110 checks to see if the byte we just previously took from the file was the last one. If our copy of ST (S) is 0, then there is still more to come so we loop back to line 80 to get more. If S is something other than 0, then it probably was the last byte and the program control passes down to line 130 which prints out a message and closes the files we opened up and ends the program. The line with "REM" right after the line numbers are ignored and are not executed. The REM command is useful for including program REMarks and notes to yourself about various things the program is about to do.

A minor variation to this program could allow you to print out the file to a printer. Add in these lines to the above program to print out to a printer as well as the screen.

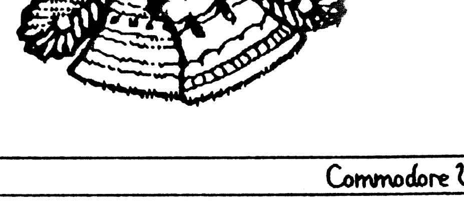
```
45 open 4,4,7
100 printa$;:print#4,a$;
140 close2:close4:close15
```

Line 45 opens up a channel (4) to the printer (4) and sets a secondary address to 7. The secondary address has different meanings to different devices. In this case, if you have a Commodore printer plugged in, the number 7 will put the printer into lowercase-uppercase. Check your printer manual for the proper secondary address number. Line 100 prints out the byte to the screen as well as to the printer. Line 140 simply closes the file. The ";" after the variable A$ seems rather innocent but does a very important function. Without it, the computer screen (or printer) would print the characters one on top of the other with each character taking up the whole line

eg. Something that should read "HELLO" would be printed out:

```
H
E
L
L
O
```

Well, next month, I will discuss creating a program that uses a seq file to store a bowling league database.



CompuServe Commodore Tricks

Hello, Commodore programmers! This file will show you some interesting SYS, POKE and PRINT features that will create some helpful (and neat) stuff on your 64!

Ready? Okay, let's get to it!

POKE 808,225
The above command will disable the RUN/STOP key and the LIST command. POKE 808,237 re-enables them. If it doesn't re-enable them, try PRINT PEEK (808) when you FIRST turn the power on. The number you get is the one to be POKEd into 808 to re-enable the RUN/STOP key and LIST command after you disable them.

POKE 650,128
Makes characters repeat if that key is held down.

PRINT PEEK (56321)
The above command tells you which direction the joystick in Port 1 is being moved. The number you get means the following:

254 - UP	253 - DOWN
251 - LEFT	247 - RIGHT
250 - UP/LEFT	246 - UP/RIGHT
249 - DN/LEFT	245 - DN/RIGHT
239 - Fire button	

For coding a certain sentence,

POKE 53272,N
This makes the characters on the screen become strange and mystic. There are 100 different sets of these "weird" characters, depending on what value you put for N. (N is any number from 0 to 99.)

POKE 53272,21 to get back to the normal letters (A B C D E F etc.) instead of the strange designs.

Finally:

FOR T=55296 TO 56295:POKE T,N:NEXT T
The above command turns all of the characters on the screen whatever color you want (Color = Value of N).

For the different color values to put in place of N, see the top of page 139 in the Commodore 64 User's Guide that comes with the computer.

I'm sure most of you know this, but for a BASIC beginner that doesn't know a whole lot yet, the command below changes the screen (background) and border colors to whatever color's value you put for N (see the top of page 139 in the C-64 User's Guide.)

Border: POKE 53280, N Screen (Background): POKE 53281, N

... more CompuServe tricks

The following programs and tips are examples of ways to make your BASIC programs look more structured.

First of all it is always best if you are writing a large program to break it up into several subroutines. These subroutines could make use of variables so as to function differently depending on what part of the program is calling them.

To give your subroutine an apparently REMless title do the following:

1) Enter the REM on desired line followed by a quote (").
2) Press <CR>
3) Cursor up to that line and to the right of that quote and turn on reverse mode (CTRL RVS-ON).
4) Now press SHIFT M and SHIFT Q.
5) Press the quote again then delete it.
6) Enter the comment you wish to be entered as a REMark.
7) Press quote and then delete it.
8) RVS-ON.
9) Next SHIFT Z.
10) Finally, <CR>.

That's all there is to it! I can't take credit for this one. It belongs to Luis Pistoia of Argentina.

To produce the indentation effect found in certain PASCAL and C editors, use the colon (:). That's all! The following give examples of what I mean:

PROGRAM #1

```
10 :FOR I = 1 TO 10
12 :   FOR J = 1 TO 10
14 :      FOR K = 1 TO 10
16 :         PRINT
18 :         PRINT
20 :         PRINT
22 :         PRINT K,
24 :         PRINT J,
26 :         PRINT I
28 :         PRINT
30 :         PRINT
32 :         PRINT
34 :      NEXT K
36 :   NEXT J
38 :NEXT I
```

PROGRAM #2

```
10 :REM   THIS IS AN EXAMPLE IN
12 :REM   STRUCTURED PROGRAM DESIGN
14 :REM
16 :BEGINNING=1: FINISH=12
18 :FOR LOOP = BEGINNING TO FINISH
20 :   PRINT "LOOP #"; LOOP
22 :   PRINT: PRINT: PRINT
24 :NEXT LOOP
99 :STOP
```

Also, in PROGRAM #2 I use long variable names to make the program more readable. Naturally BASIC will only read the first two characters of the variable. For example the BASIC interpreter will refer to the variable BEGINNING as BE. (Editor's note: this program won't work successfully because of the imbedded tokens in the variable names, such as the IN in BEGINNING. Care should be taken not to include BASIC keywords in long variable names. -jbc)

I hope these hints are helpful. If they are, pass them on!

EXPERTS LIST

Wordprocessing		
Paperclip (to version E)	Jarrett Currie	757 2391
Paperclip (any version)	Ken Danylczuk	545 0644
Pocket Writer	Barry Bircher	543 8840
Pocket Writer	Real Charron	586 1843
Spreadsheets		
Pocket Planner	Barry Bircher	543 8840
Better Working SS	Ken Danylczuk	545 0644
Databases		
Pocket Filer	Barry Bircher	543 8840
Oracle (Consultant)	Ken Danylczuk	545 0644
Communication		
Desterm 2.0	Barry Bircher	543 8840
Dialogue 128	Jarrett Currie	757 2391
Library files	Barry Bircher	543 8840
Music/Sound		
(Most)	Ken Danylczuk	545 0644
Languages		
Forth	Ken Danylczuk	545 0644
Pascal	Ken Danylczuk	545 0644
ML (machine language)	Ken Danylczuk	545 0644
ML (machine language)	Barry Bircher	543 8840
BASIC (2.0-7.0, files)	Ken Danylczuk	545 0644
Graphics		
Print Shop/Master	Ken Danylczuk	545 0644
Koala Painter/Printer	Ken Danylczuk	545 0644
Hardware		
Disk Drive Maintenance	Ken Danylczuk	545 0644
GEOS		
GEOS 64	Jarrett Currie	757 2391
GEOPublish 64	Jarrett Currie	757 2391
GEOS 128	Barry Bircher	543 8840
CP/M		
CP/M 128	Jarrett Currie	757 2391
Archiving Programs	Jarrett Currie	757 2391