I/O Routines

The I/O routines for BASIC include the BASIC commands OPEN, CLOSE, SAVE, LOAD, VERIFY, INPUT#, GET#, CMD, and PRINT#.

CMD, PRINT#, GET#, and INPUT# require a previous OPEN so that these commands can relate their logical file number parameter to the corresponding entry in the device number table.

OPEN from BASIC calls the Kernal SETLFS routine to specify the logical file number, device number, and secondary address for a file. The OPEN information for these three items is stored in three tables that can each contain ten entries. One table is for logical file numbers, another for device numbers, and a third for secondary addresses.

The three values for any particular file are stored in the same relative-order in the three files. When one of the I/O commands requests a particular logical file number, that file number is searched for in the logical file number table. If it is found, the corresponding entries in the device number and secondary address tables can be located.

If the logical file number isn't found, the FILE NOT FOUND error occurs. If you try to OPEN a file and ten logical files have already been OPENed, you get the TOO MANY FILES error. It is not possible to OPEN a logical file number zero.

OPEN also does certain other things besides managing the logical file number, device number, and secondary address tables. For a serial device, OPEN commands the serial device to listen and then sends a secondary address for OPEN to this serial device. The filename information can also be sent to the serial device. For tape, OPEN checks for a tape header of a sequential file if reading or writes a tape header for a sequential file if writing to tape. For RS-232 certain handshake procedures are performed, and two 256-byte buffers are stolen from the top-of-BASIC memory to provide room for the buffer to transmit and receive characters.

Once the OPEN is completed, it is possible to do I/O to a logical file number that corresponds to a logical file that is open. Simply specify an INPUT#, CMD, PRINT#, or GET#,

and BASIC handles the rest. BASIC calls the appropriate Kernal I/O routines as described below. BASIC itself performs no direct I/O operations. For details about the Kernal I/O routines, see the Kernal volume of *The Tool Kit.*

First a channel to the device is established. The CMD (and PRINT# which calls CMD) calls the Kernal CHKOUT routine to open an output channel. For tape, this involves checking the secondary address for a valid write specification and displaying NOT OUTPUT FILE if you gave a secondary address for READ. For serial and RS-232 devices, handshake procedures are performed. If the serial device doesn't handshake as expected, the DEVICE NOT PRESENT error message occurs. For INPUT# and GET#, the Kernal CHKIN routine to open an input channel is called. If the current input device for this logical file has a secondary address for a write and you try to do INPUT# or GET#, the NOT INPUT FILE error occurs. The handshake is performed for RS-232 or serial devices and the DEVICE NOT PRESENT can occur for a serial device.

Now that the channel is established, you can start transmitting or receiving characters over it. The Kernal CHROUT routine is used by PRINT. Unless you change things through an OPEN, the default channel is the screen, and that's where PRINT normally sends its characters. For other devices, CMD and PRINT# call PRINT to use this CHROUT routine.
The Kernal CHRIN routine is used by INPUT# to fill the BASIC input buffer at 0200. This same CHRIN routine is also used to fill the buffer when you are just entering characters on the screen. The Kernal GETIN routine is called by GET or GET# to receive one character from the current input device. GETIN doesn't have the possibility of hanging the system like CHRIN does since GETIN always returns with a $00 if it retrieves a null character or with a $0D if a serial I/O error occurs.

One feature of BASIC that is at times confusing is the difference between PRINT# and CMD. Also, how are LIST and PRINT related to CMD? Below is a short list which should help summarize the relationships:

PRINT#: Calls CMD; Calls CLRCHN to close all I/O channels and reset current output device to be the screen.
CMD: Calls CHKOUT to set new current output device; Calls PRINT.
PRINT: Sends characters to the current output device.

LIST: Calls PRINT.

The INPUT# and GET# commands are discussed in the section on INPUT/GET/READ. All of the other I/O commands are discussed in this section.

SAVE and LOAD/VERIFY both use the Kernal SAVE and LOAD/VERIFY routines to do all of the I/O work. The SETLFS and SETNAM that are required for the Kernal SAVE and LOAD/VERIFY are done by the BASIC SAVE and LOAD/VERIFY commands when these commands parse the parameters following the command.

One of the features for SAVEing to tape that is frequently specified incorrectly is the information about the secondary address. So far neither the VIC-20 nor Commodore 64 *Programmer's Reference Guide* has the information all straight. One source that does is Russ Davies' *Mapping the VIC.* Here's what we found for the tape SAVE secondary address: First, any even secondary address results in a tape ID of 1 to produce a relocatable program tape (examples: $00, $02, $04). (Relocatable means it can be loaded back into a different area of memory than the area it was saved from.) Any odd secondary address gives a tape ID of 3 for a nonrelocatable program tape (examples: $01, $03, $05). Also, if you have bit 1 on (the second bit from the right) in the secondary address, an end-of-tape header with a tape ID of 5 is written following the program (examples: $02, $03). If you don't specify anything for a secondary address for SAVE, a default of $00 is used, thus producing a relocatable program tape. This is the kind of tape that you should create when saving BASIC programs so that they can be loaded back to wherever BASIC currently finds the area that is reserved for its tokenized program. So my advice is: Leave the secondary address alone and everything works fine.

The logical file number you specify in an I/O command can be any numeric expression (including floating point and scientific notation) as long as the final integer result is in the range of 1–255.

The *Programmer's Reference Guide* says, "When PRINT# is used with tape files, the comma, instead of spacing by print zones, has the same effect as a semicolon." The *Guide* also says, "The data items are written as a continuous stream of characters." There is nothing in PRINT that check for a tape device when commas are detected. The only device specific

item with PRINT connected with spaces is that PRINT sends a cursor-right to the screen for a space and sends actual spaces to other devices. When I tried PRINT# to a tape file using a list of variables separated by commas and then read the tape file back in using GET#, I found that the spaces were indeed sent just like PRINT was tabbing to the next TAB position for a comma.

One of the pleasant surprises when investigating the tape SAVE was that the restriction against SAVEing from $8000 or above that had existed on the VIC-20 has been removed on the Commodore 64. This fix was done by eliminating the check in tape SAVE that had used the same byte both for an indication that the SAVE was done (high-order bit on) and for the high-order byte of the area being SAVEd (when it reached $80 it forced an end-of-SAVE). Changing this limitation was a necessity on the Commodore 64 since the default BASIC memory area goes past $8000. If this change had not been made, any very large programs that went past $8000 couldn't have been completely saved.

OPEN
E1BE/E1BB-E1C6/E1C3

Called by: Execute the Current BASIC Statement for OPEN.

Operation:
1. JSR E219/E216 to handle the parameters for OPEN.
2. JSR FFC0 to the Kernal OPEN routine to open this logical file to allow use of the file in CMD, PRINT#, GET#, or INPUT#.
3. If an error occurred during OPEN, branch to E1D1/E1CE which jumps to E0F9/E0F6.

CLOSE
E1C7/E1C4-E1D3/E1D0

Called by: Execute the Current BASIC statement for CLOSE.

Operation:
1. JSR E219/E216 to handle the parameters for CLOSE, leaving the file number in 49.
2. Load the accumulator with this file number from 49.
3. JSR FFC3 to the Kernal CLOSE routine to close this logical

file, removing the entries for it from the logical file number, device number, and secondary address tables.
4. If there are no errors during CLOSE, exit.
5. If an error occurred during CLOSE or if it's a CLOSE for an RS-232 device, JMP E0F9/E0F6.

Handle Parameters for OPEN/CLOSE E219/E216-E263/E260

Called by: JSR at E1BE/E1BB OPEN; JSR at E1C7/E1C4 CLOSE.

Operation:
1. Load the accumulator with $00. JSR FFBD to the Kernal SETNAM routine to set a default filename size of zero in B7.
2. JSR E211/E20E to see if any parameters follow the OPEN/CLOSE. If no parameters, display SYNTAX ERROR.
3. JSR B/D79E to evaluate the first parameter following the OPEN/CLOSE. If it's nonnumeric, display TYPE MISMATCH. If it's not in the range 0–255, display ILLEGAL QUANTITY. Return with the value in the X-register.
4. Store the X-register in 49 to save the file number in case more than one parameter is present. Also transfer the X-register to the accumulator.
5. LDX $01 to set default device number of 1, the tape cassette.
6. LDY $00 to set default secondary address of zero for tape READ.
7. JSR FFBA to the Kernal SETLFS routine to set the logical file number from the accumulator, the device number from the X-register, and the secondary address from the Y-register.
8. JSR E206/E203 to see if the end of the OPEN/CLOSE statement has been reached. If so, just exit after processing this parameter for the file number.
9. JSR E200/E1FD to check for a comma following the first parameter. If not found, display SYNTAX ERROR. If a comma is found, evaluate the second parameter. Again return the value in the X-register of 0–255 and display TYPE MISMATCH if it's a string expression or ILLEGAL QUANTITY if it's not 0–255.

10. Store the X-register in 4A to save the device number in case more than two parameters are present.
11. LDY $00 to set a default secondary address of zero for read.
12. LDA 49 to retrieve the first parameter that gave the file number.
13. If 4A, the device number, is greater than three and thus indicating a serial device, DEY to set secondary address of $FF.
14. JSR FFBA to the Kernal SETLFS routine to set the logical file number from the accumulator, the device number from the X-register, and the secondary address from the Y-register.
15. JSR E206/E203 to see if the end of the OPEN/CLOSE statement has been reached. If so, just exit after processing these first two parameters for the file number and the device.
16. JSR E200/E1FD to check for a comma following the second parameter. If not found, display SYNTAX ERROR. If a comma is found, evaluate the third parameter. Again return the value in the X-register of 0–255 and display TYPE MISMATCH if it's a string expression or ILLEGAL QUANTITY if it's not 0–255.
17. TXA and TAY to move this third parameter to the secondary address register.
18. LDX 4A to retrieve the device number previously set.
19. LDA 49 to retrieve the logical file number.
20. JSR FFBA to the Kernal SETLFS routine to set the logical file number from the accumulator, the device number from the X-register, and the secondary address from the Y-register.
21. JSR E206/E203 to see if the end of the OPEN/CLOSE statement has been reached. If so, just exit after processing these first three parameters for the file number, the device, and the secondary address.
22. JSR E20E/E20B to check for a comma following the third parameter. If not found, display SYNTAX ERROR.
23. If a comma is found, evaluate the fourth parameter by JSR A/CD9E. Return with (64) pointing to the string descriptor for this fourth parameter, the filename. This fourth parameter contains the filename, which can also include the type such as SEQ and the mode such as W.

24. JSR B/D6A3 to see if the expression from step 23, the fourth parameter, was a string. If not, display TYPE MISMATCH. If it was a string, remove the string descriptor from the temporary string descriptor stack with (22) pointing to the string and the accumulator containing its length.
25. LDX from 22 and LDY from 23.
26. JMP FFBD to the Kernal SETNAM routine to set the filename using the pointer to the name in the X-register and Y-register and the length of the name from the accumulator.

Check for Comma in Parameter List and Evaluate Following Parameter to Integer Value in X-Register E200/E1FD-E205/E202

Called by:
JSR at E1E9/E1E6, E1F6/E1F3 LOAD/SAVE Parameters; JSR at E231/E22E, E245/E242 OPEN/CLOSE Parameters.

Operation:
1. JSR E20E/E20B to check for a comma and display SYNTAX ERROR if it's not found.
2. JMP B/D79E to evaluate the expression following the comma. If it's nonnumeric, display TYPE MISMATCH. If it's not 0–255, display ILLEGAL QUANTITY.

Check If End of Parameter List E206/E203-E20D/E20A

Called by:
JSR at E1E0/E1DD, E1E6/E1E3, E1F3/E1F0 LOAD/SAVE Parameters; JSR at E22E/E22B, E242/E23F, E251/E24E OPEN/CLOSE Parameters.

Operation:
1. JSR CHRGOT to load the accumulator with the last character scanned in the parameter list.
2. If not the $00 end-of-line or colon end-of-statement, then RTS.
3. If end-of-line or end-of-statement, pull this routine's return address off the stack and then RTS. Thus this RTS will serve as the RTS for the LOAD/SAVE Parameters Routine or the OPEN/CLOSE Parameters Routine.

Check for Comma Between Parameters E20E/E20B-E218/E215

Called by:
JSR at E200/E1FD Evaluate Following Parameter to Integer; JSR at E254/E251 OPEN/CLOSE Parameters; Alternate E211/E20E JSR at E21E/E21B OPEN/CLOSE Parameters.

Operation:
1. JSR A/CEFD to check for a comma and display SYNTAX ERROR if not found. A/CEFD then calls CHRGET to retrieve the following nonspace character.
2. E211/E20E: JSR CHRGOT to load this last character retrieved back into the accumulator.
3. If it's not zero or colon, then RTS.
4. If it's the $00 end-of-line or colon end-of-statement, display SYNTAX ERROR since a comma appears without a following parameter.

SAVE E156/E153-E164/E161

Called by: Execute the Current BASIC Statement for SAVE.

Operation:
1. JSR E1D4/E1D1 to handle the SETNAM and SETLFS procedures for SAVE.
2. Set the end address plus one for the SAVE. Load the X-register from 2D. Load the Y-register from 2E. (2D) points to the end of the tokenized program plus one.
3. Set the starting address for the SAVE. LDA $2B to set the offset to (2B). (2B) contains the starting address of the tokenized program.
4. JSR FFD8 to the Kernal SAVE routine.
5. If the carry is set upon return, an error occurred during SAVE, so branch to E0F9/E0F6 to handle the error.
6. If the carry is clear, the SAVE did not have any errors, so exit.

VERIFY E165/E162-E167/E164

Called by: Execute the Current BASIC Statement for VERIFY.

Operation:
Load the accumulator with $01 to indicate that this is a VERIFY operation, and then fall through to the LOAD routine, step 2, through the use of a BIT instruction.

LOAD
E168/E165-E1BD/E1BA

Called by:
Execute the Current BASIC Statement for LOAD; Alternate E167/E164: Fall through from VERIFY.

Operation:
1. Load the accumulator with $00 to indicate LOAD.
2. E167/E164: Store the accumulator into 0A, which is $00 for LOAD or $01 for VERIFY.
3. JSR E1D4/E1D1 to do the SETNAM and SETLFS for LOAD/VERIFY.
4. Load the accumulator from 0A. LDX from 2B. LDY from 2C. (2B) points to the start of the area for a tokenized BASIC program.
5. JSR FFD5 to the Kernal LOAD/VERIFY routine.
6. If the carry is set upon return, branch to E1D1/E1CE to handle the error.
7. If there's no error, load the accumulator from 0A.
8. If the accumulator is zero, a LOAD was done, so branch to step 13.
9. For VERIFY: JSR FFB7 to the Kernal READST routine to read the I/O status.
10. If there are no I/O errors, branch to step 12.
11. For any I/O errors during VERIFY, display VERIFY ERROR.
12. If the VERIFY was done from program mode, just exit. If it was done from direct mode, display OK and exit.
13. For LOAD: JSR FFB7 to the Kernal READST routine to read the I/O error byte into the accumulator.
14. AND the accumulator with $BF. Thus the end-of-file for cassette READ error is not detected.
15. If the accumulator is now zero, branch to step 17.
16. For the accumulator of nonzero, display LOAD ERROR and exit to the Main·BASIC Loop.
17. If LOAD was issued from program mode, branch to step 21.

18. For a LOAD from direct mode: Store the X-register in 2D and the Y-register in 2E. (2D) points to the end of the BASIC program plus one.
19. Display READY.
20. JMP A/C52A to reset (7A) to the start of the BASIC program, relink BASIC lines, and go to the Main BASIC Loop.
21. For a program mode LOAD: JSR A/C68E to reset (7A) from the (2B) pointer to the start of the BASIC program.
22. JMP A/C533 to relink the BASIC lines.
23. JMP A/C677 to restore DATA pointer to the start minus one of the text area, reset the stack pointer, reset the temporary string descriptor stack pointer, disallow CONT, and clear the flag that prevents subscripts. The pointers to the variable area, the arrays, the free area, and the active string area are not reset, so all variables are still retained from the previous program. If the program that is LOADed is longer than the original program, the newly LOADed longer program will overlay the variable area.

    Note: Steps 22 and 23 are found at E476 on the VIC-20 in a patch area that was put back into the section for LOAD on the Commodore 64.

Handle LOAD/SAVE Parameters for SETNAM and SETLFS
E1D4/E1D1-E1FF/E1FC

Called by:
JSR at E156/E153 SAVE; JSR at E16C/E169 LOAD/VERIFY.

Operation:
1. Load the accumulator with $00 and JSR FFBD to the Kernal SETNAM routine to set a default filename of length zero for null filename.
2. LDX $01 to set the default device number. LDY $00 to set default secondary address for READ (LOAD).
3. JSR FFBA to the Kernal SETLFS routine to set the default secondary address and device number.
4. JSR E206/E203 to see if there are any parameters for LOAD/SAVE. If not, just exit.
5. JSR E257/E254 to evaluate the first parameter, the filename. If not a string, display TYPE MISMATCH. Set the accumulator to the length of the name, the X-register

and the Y-register to its location, and then use the Kernal SETNAM routine to use this filename for the LOAD/SAVE.

6. JSR E206/E203 to see if any parameter follows the filename. If not, just exit.
7. JSR E200/E1FD to check for a comma following the filename. If none is found, display SYNTAX ERROR. After the comma, evaluate the second parameter, the device number. If this second parameter is a string, display TYPE MISMATCH. If it's not in the range 0–255, display ILLEGAL QUANTITY. Return with the number in the X-register.
8. Load the Y-register with $00 to set the default secondary address for READ.
9. STX in 49 to save the device number in case a third parameter is present.
10. JSR FFBA to the Kernal SETLFS routine to use this device number just specified and the secondary address of zero.
11. JSR E206/E203 to see if any parameter follows the device number. If not, just exit.
12. JSR E200/E1FD to check for a comma following the device number. If none is found, display SYNTAX ERROR. After the comma, evaluate the third parameter, the secondary address. If this third parameter is a string, display TYPE MISMATCH. If it's not in the range 0–255, display ILLEGAL QUANTITY. Return with the number in the X-register.
13. LDX from 49, the same device number parameter.
14. JMP FFBA to the Kernal SETLFS routine.

PRINT#
A/CA80–A/CA85

Called by: Execute the Current BASIC Statement for PRINT#.

Operation:

1. JSR A/CA86 to execute CMD, which opens a channel for output for the logical file specified and then executes PRINT.
2. JMP A/CBB5 to call the Kernal CLRCHN routine to clear all channels and reset 13, the I/O command file number, to zero.

CMD
A/CA86-A/CA99

Called by:
Execute the Current BASIC Statement for CMD; JSR at A/CA80 PRINT#.

Operation:
1. JSR B/D79E to convert the expression following the CMD or PRINT#, which should be the file number of a previously opened file, into an integer from 0 to 255. If it's out of this range, display ILLEGAL QUANTITY. If it's a string expression, display TYPE MISMATCH. Return with the integer in the X-register.
2. If nothing followed the file number, branch to step 4.
3. If something did follow the file number, see if this character was a command. If not, display SYNTAX ERROR.
4. PHP to save the BEQ status if nothing followed the file number. PRINT uses this status information to know if a blank line is to be printed.
5. STX in 13, the I/O file number for PRINT and LIST.
6. JSR E118/E115 to execute the Kernal CHKOUT routine to open the channel for this logical file number.
7. PLP to restore a possible blank line status indicator.
8. JMP A/CAA0 to execute PRINT. If no parameters followed the file number, PRINT just sends a carriage return. If the file number is >=128, PRINT also sends linefeed.

BASIC's CHROUT
E10C/E109-E111/E10E

Called by: JSR at A/CB47 PRINT.

Operation:
1. JSR FFD2 to the Kernal CHROUT routine to send the character in the accumulator to the current output device, 9A.
2. If carry is set, branch to E0F9/E0F6 to handle Kernal call errors.

BASIC's CHRIN
E112/E10F-E117/E114

Called by:
JSR at A/C562 Receive Characters and Fill BASIC Input Buffer.

Operation:
1. JSR FFCF to the Kernal CHRIN routine to load a character into the accumulator from the current input device, 9A.
2. If carry is set, branch to E0F9/E0F6 to handle Kernal call errors.

BASIC's CHKOUT
E118/E115-E11D/E11A

Called by: JSR at A/CA93 CMD.

Operation:
1. JSR FFC9 to the Kernal CHKOUT routine.
2. If carry is set, branch to E0F9/E0F6 to handle Kernal call errors.

BASIC's CHKIN
E11E/E11B-E123/E120

Called by: JSR at A/CB8F GET#; JSR at A/CBAF INPUT#.

Operation:
1. JSR FFC6 to the Kernal CHKIN routine.
2. If carry is set, branch to E0F9/E0F6 to handle Kernal call errors.

BASIC's GETIN
E124/E121-E129/E126

Called by: JSR at A/CC35 GET/GET#.

Operation:
1. JSR FFE4 to the Kernal GETIN routine.
2. If carry is set, branch to E0F9/E0F6 to handle Kernal call errors.

Handle Kernal Call Errors and RS-232 CLOSE E0F9/E0F6-E10B/E109

Called by:
The BASIC Kernal Call Routines shown above and by a JMP at E1D1/E1CE for OPEN/CLOSE.

Operation:
1. Compare the accumulator to $F0, which is the value set by the RS-232 CLOSE routine.
2. If not equal, branch to step 5.
3. For an RS-232 CLOSE, STY 38 and STX 37 to reset the pointer to the end of BASIC memory since the RS-232 buffers have been released.
4. JMP A/C663 to execute CLR.
5. Transfer the accumulator to the X-register to be used to retrieve an error message.
6. JMP A/C437 to display the error message and return to the Main BASIC Loop.