Simon’s Mostly Reliable Guide to KERNAL I/O

Table of Contents

Introduction

This document tries to explain the “why”, “what” and “how” details of input/output handling by Commodore 8-bit computers from the VIC-20 onwards. The specifics mentioned apply to the VIC-20, it is possible they vary slightly with other models.

In contrast to many other systems of the era Commodore designed a flexible and versatile interface to allow programmers to interact with peripherals and to enable new hardware devices to be integrated without major changes to applications.

BASIC programs can access printers, disk drives and other devices using high-level concepts with a set of simple commands. Machine code programs also have a well-defined and documented set of KERNAL routines which perform the equivalent functions.

Terminology

I will use the same terms commonly referred to in Commodore documentation and other reference material. These will be shown in *italics* when they are first mentioned.

Logical File

Both the computer and peripheral need a handle to refer to an interaction between each other, there may be multiple instances to different devices or the same device at any one time. For the computer this reference is called a *logical file*.

A logical file has a defined lifecycle: it is opened, I/O is performed and then it is closed. The number of logical files that can be active at any one time is finite, the computer can have at most ten logical files open and many devices have a lower limit. A logical file might not be associated with an actual file on a device such as a disk drive, for example a logical file is also used for sending output to a printer.

The computer uses a *logical file number* (LFN) to identify to a specific logical file. An LFN is an integer between 1 and 255.

Devices are not aware of LFNs and have their own internal mechanisms to identify between logical files.

Device Number

Each logical file must be associated with a specific target device. The following table lists typical *device numbers*:

Device Number	*Device Type*
0	Keyboard
1	Cassette
2	RS-232 port
3	Screen
4, 5	Printer
6, 7	Plotter
8-11	Disk drive
12-30	Other serial device

Device numbers of 4 and above are external to the computer, normally on the serial bus.

Secondary Address

For a device to differentiate between two logical files, or to select different behaviour, a logical file may have a *secondary address* (SA). The purpose and valid values of a secondary address vary between devices, full details will be given in the specific section for the device below.

In some circumstances the computer also treats a logical file differently depending on secondary address, again this is explained below.

I/O with BASIC

Commodore BASIC has a number of commands and statements to perform I/O operations.

Opening a Logical File

The `OPEN` statement is used to open a new logical file. In addition to LFN, device number and SA a variable length string (often a file name or command) may be given.

```
OPEN <LFN>,<device number>[,<SA>[,<file name>]]
```

Attempting to reuse an open LFN will result in a ?FILE OPEN ERROR. Other errors may not be indicated until read or write operations are attempted.

Output to a Logical File

Data is sent to a logical file using the `PRINT#` statement.

```
PRINT#<LFN>,<value1>[;<valueN>]
```

As with the `PRINT` statement a Carriage Return will be appended unless the statement ends with a semicolon (;).

Attempting to write to a logical file that is not open will result in a ?FILE NOT OPEN ERROR. Other errors may result in the `ST` variable being set.

Another statement which controls output is

```
CMD <LFN>[,<string>]
```

All further output using the `PRINT` statement will be sent to the logical file specified instead of the screen. In this way output from a program or a listing can be sent to a device (such as a printer).

To flush output the statement

```
PRINT#<LFN>
```

should be made before closing the logical file.

LFNs of 128 and above have special behaviour with respect to line endings. After every Carriage Return ($0D) a Line Feed ($0A) is also sent.

Input from a Logical File

Two statements are available to read data from a logical file.

For line-based information the `INPUT#` statement reads one or more records into variables

```
INPUT#<LFN>,<variable1>[,variableN]
```

Each line is is terminated by a Carriage Return. Each record is separated from the next by any of the following characters

- comma (,)
- semicolon (;)
- colon (:)

In order to have a record containing any of these characters the record must be written surrounded by double quotes ("). A record cannot contain a Carriage Return.

If the destination for a record is a numeric variable the record must be a valid floating point number, otherwise a ?FILE DATA ERROR is reported.

Single bytes can be read from a logical file using the `GET#` statement

```
GET#<LFN>,<variable1>[,variableN]
```

Single characters are read and assigned to each variable, this includes the record separators listed above and Carriage Return. If the destination is a numeric variable the character must be a decimal digit otherwise a ?FILE DATA ERROR is reported. If a NUL ($00) byte is read using `GET#` it will result in an empty string being assigned to the variable.

When the last byte of data for a logical file is read bit 6 of the status variable `ST` is set to indicate end of file.

Neither `INPUT#` nor `GET#` may be executed in direct mode, they can only be used within a BASIC program.

Closing a Logical File

When all I/O operations on a logical file have been performed it must be closed

```
CLOSE <LFN>
```

Without an explicit close data written to a logical file may be lost.

Saving a Program

The current BASIC program can be saved to a device using the `SAVE` command

```
SAVE [<file name>[,<device number>[,<SA>]]]
```

The only valid devices to save to are cassette and disk. If no device number is given the program will be saved to cassette. If no secondary address is given 0 is used.

Loading a Program

A program (either BASIC or machine code) can be loaded into memory using the `LOAD` command

```
LOAD [<file name>[,<device number>[,<SA>]]]
```

The only valid devices to load from are cassette and disk. If no device number is given the program will be loaded from cassette. If no secondary address is given 0 is used.

Checking Operation Status

To signal non-fatal errors and other events the status variable `ST` may be read. The value returned is a set of bits, the meaning of each bit depends on what device was the target of the last I/O operation.

The `ST` variable may not be assigned to.

I/O with Machine Code

Machine code programs perform I/O using a set of KERNAL routines, parameters are passed in registers and zero page locations, results are returned using the same.

Unless otherwise noted errors are indicated by the carry bit of the Status Register (Cb) being set and one of the following values in the Accumulator (.A)

Value	*Description*	*Value*	*Description*
0	STOP key pressed	5	Device not found
1	Too many open files	6	File is not an input file
2	File already open	7	File is not an output file
3	File not open	8	File name is missing
4	File not found	9	Illegal device number

Some routines may update the `STATUS` ($90) byte which should be read using the `READST` routine.

Opening a Logical File

Three routines must be called to open a new logical file, the first two can be called in either order.

SETLFS ($FFBA)
Set logical file number, device number and secondary address

IN		OUT
.A	LFN	–
.X	Device number	
.Y	SA	

Store the parameters needed for a new logical file.

SETNAM ($FFBD)
Set file name

IN		OUT
.A	File name length	–
.X	File name address low byte	
.Y	File name address high byte	

Store the file name for a new logical file. If no file name is required just the length needs to be set (to zero).

OPEN ($FFC0)
Open logical file

IN	OUT	
–	.A	Error code
	ST	

On success Cb will be clear. Depending on device errors may be reported in `STATUS`.

Output to a Logical File

The KERNAL has an *output channel* that is used for all output operations. By default this is assigned to the screen (device 3). Output can be directed to the device associated with a logical file using the `CHKOUT` routine.

CHKOUT ($FFC9) Open channel for output			
IN		**OUT**	
.X	LFN	.A	Error code

On success Cb will be clear and all future output will be sent to the logical file until the `CLRCHN` routine is called.

Data is sent to the output channel using the `CHROUT` routine.

CHROUT ($FFD2) Send byte to output channel			
IN		**OUT**	
.A	Byte to send	.A	Error code
		ST	

Input from a Logical File

The KERNAL has an *input channel* that is used for all input operations. By default this is assigned to the keyboard (device 0). Input can be directed to the device associated with a logical file using the `CHKIN` routine.

CHKIN ($FFC6) Open channel for input			
IN		**OUT**	
.X	LFN	.A	Error code

On success Cb will be clear and all future input will be taken from the logical file until the `CLRCHN` routine is called.

For most devices data is read from the input channel using the `CHRIN` routine.

CHRIN ($FFCF) Read byte from input channel			
IN		**OUT**	
–		.A	Data
		ST	

The routine blocks until data is available or a permanent error is detected.

The `GETIN` routine allows a program to try to read from the keyboard or RS-232 port without blocking.

GETIN ($FFE4) Check for byte from input channel			
IN		**OUT**	
–		.A	Data

If the Cb is clear then .A contains the data byte, otherwise no data is available.

Calling `GETIN` for other devices has the same behaviour as `CHRIN`.

Clearing Channels

In order to switch either the input or output channel to another logical file, or to restore it to its default source the `CLRCHN` routine must be used.

CLRCHN ($FFCC) Clear channels			
IN		**OUT**	
–		ST	

Channels must be cleared before a logical file is closed.

Closing a Logical File

When all I/O operations on a logical file have been performed it must be closed by calling the `CLOSE` routine.

CLOSE ($FFC3) Close logical file			
IN		**OUT**	
.A	LFN	.A	Error code

Without an explicit close data written to a logical file may be lost.

The KERNAL also has a `CLALL` routine, this discards all open logical files and clears the input and output channels but does not notify external devices to allow them to close any logical files.

Saving a Program

A program (or any region of memory) can be saved to a device using the `SAVE` routine. Before calling it the `SETLFS` and `SETNAM` routines must be called. The LFN passed to `SETLFS` is not used.

The start address must be stored in two consecutive zero page addresses and the Accumulator loaded with the address of the first byte. Memory is saved up to, but not including the end address.

SAVE ($FFD8) Save file to device			
IN		**OUT**	
.A	Zero page with start address	.A	Error code
.X	End address low byte	ST	
.Y	End address high byte		

The only valid devices to save to are cassette and disk.

Loading a Program

A program can be loaded from a device using the `LOAD` routine. Before calling it the `SETLFS` and `SETNAM` routines must be called. The LFN passed to `SETLFS` is not used.

LOAD ($FFD5) Load file from device			
IN		**OUT**	
.A	0 – load, !0 – verify	.A	Error code
.X	Load address low byte	.X	End address low byte
.Y	Load address high byte	.Y	End address high byte
		ST	

The end address points to the location immediately after the last byte loaded.

The only valid devices to load from are cassette and disk.

Checking Operation Status

To signal errors and other events the status may be read using the `READST` routine.

READST ($FFB7) Read status			
IN		**OUT**	
	–	.A	Status

The value returned is a set of bits, the meaning of each bit depends on what device was the target of the last I/O operation.

Device-specific Behaviour

Operations on logical files behave slightly differently depending on the target device. This section covers how commands and routines operate for each type of device.

Keyboard

Only input operations can be performed on the keyboard.

Calling the `CHRIN` KERNAL routine for the first time turns on the cursor and waits for a line terminated with a Carriage Return before returning the first character of the logical line entered. Subsequent calls will return the remaining characters, including the final Carriage Return.

File names and secondary addresses have no meaning for logical files associated with the keyboard.

Cassette

The cassette tape can store program files and data files with optional names. Files can only be read with the corresponding command or function it was written with, for example a file that was SAVEed must loaded with either the `LOAD` BASIC command or the `LOAD` KERNAL routine.

When searching for a file by name with `OPEN` or `LOAD` the first file whose name is equal to or is a superset of the given name will be matched. For example if the command is

```
LOAD "EXAM"
```

then either the file “EXAMPLE” or “EXAMINE” would be loaded, the file “EX” would be ignored. Wildcards are not supported so the characters “`?`” and “`*`” would have to be exact matches. If no name is given then the next file of the appropriate type (data or program) will be read.

If an End of Tape record is reached before a matching file name the operation aborts with a “Device not found” error (not, as documented, setting b7 in the status byte).

Writing Data Files

Data files are created by opening a logical file with a secondary address of 1 or 2.

Data is written in 192 byte blocks until the file is closed when any remaining data followed by a NUL ($00) byte is written out. If the logical file has a secondary address of 2 then an End of Tape record is also written.

Reading Data Files

Data files are read by opening a logical file with a secondary address of 0.

Data is read until a NUL byte is reached when the End of File bit is set in the status byte. This means cassette data files cannot contain NUL bytes (except as the final character).

Saving a Program

Program files are written as a single unit to tape. When SAVEd with a secondary address of 1 the file is written with a record type that indicates the file is not relocatable.

If a secondary address of 2 is used then an End of Tape record is written after the program.

Loading a Program

The load address of a program is determined as follows:

- If the file was saved with a non-relocatable header it is loaded to the address it was saved at
- If the file was saved with a relocatable header and the SA is 1 it is loaded to the address it was saved at
- Otherwise it is loaded to the start of BASIC (BASIC `LOAD` command) or the address provided (KERNAL `LOAD` routine).

Checking Operation Status

The status byte (`ST` variable or result of `READST` routine) has the following bits set after a cassette operation.

Bit	*Description*
2	Short block
3	Long block
4	Read error
5	Checksum error
6	End of file

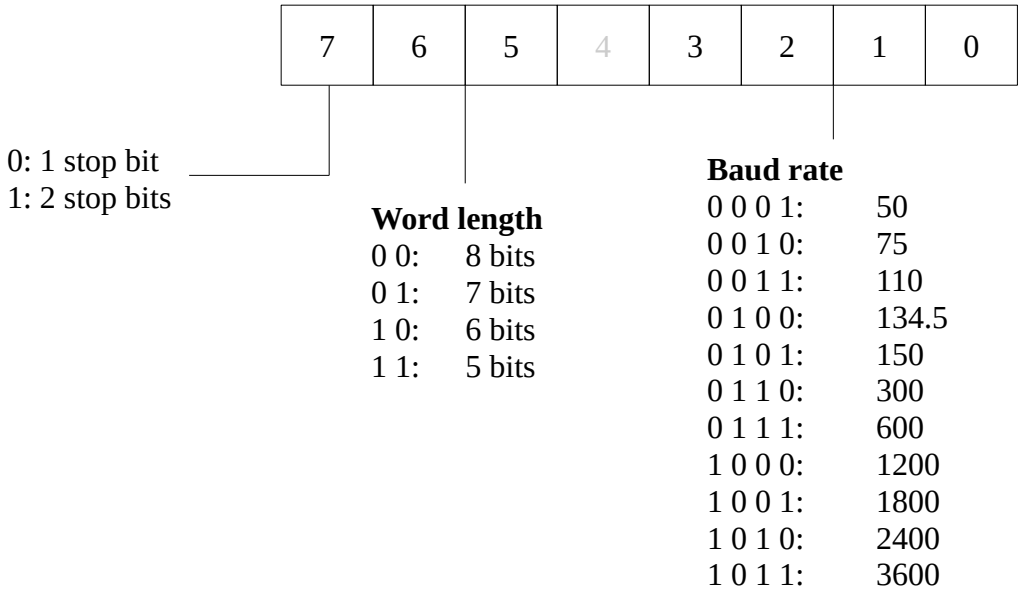
RS-232 Port

The RS-232 port provides an interface to peripherals such as printers and modems, as well as other computers.

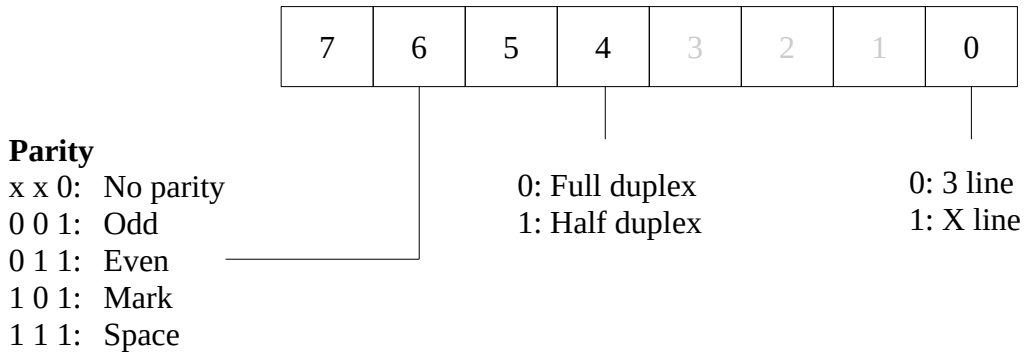
Opening a Logical File

A file name must be supplied when opening the logical file, it must be at least two bytes in length which define the control and command registers.

Control Register



Command Register



A logical file may be used for both the input channel and the output channel simultaneously.

Because of bugs in the KERNAL the X line mode should not be used.

The return value from the `OPEN` KERNAL routine always has the Cb set and the value $F0 in .A. If a logical file is opened by a BASIC program all variables are cleared.

Opening a logical file causes the top two pages of RAM to be assigned to the receive and transmit buffers.

Secondary addresses have no meaning for logical files associated with the RS-232 port. Only one logical file to the RS-232 port should be open at a time.

Closing a Logical File

When the RS-232 logical file is closed the port is immediately returned to an idle state. There is no official mechanism to tell that the transmit buffer is empty.

The return value from the `CLOSE` KERNAL routine always has the Cb set and the value $F0 in .A. If a logical file is closed by a BASIC program all variables are cleared.

Checking Operation Status

Because of a KERNAL bug checking the status byte using the `ST` variable or by calling the `READST` routine will always return 0. To determine the actual status `RSSTAT` ($297) must be read directly.

Bit	*Description*
0	Receive parity error
1	Receive framing error
2	Receive buffer overrun
4	No CTS detected
6	No DSR detected
7	Break

Screen

A logical file opened to the screen can can be assigned to the input channel. Reading from such a logical file returns characters on the current logical line using the same rules as given in the relevant “Input from a Logical File” section above.

File names and secondary addresses have no meaning for logical files associated with the screen.

Printer

Only output operations can be performed to printers.

Different models support various control character sequences and secondary addresses. A secondary address that is supported by many printers is 7 which selects business mode (upper & lower case characters).

File names have no meaning for logical files associated with the printer.

Plotter

Only output operations can be performed to plotters.

The 1520 plotter uses secondary addresses to select different modes and functions.

SA	*Description*
0	Print ASCII data
1	Plot X, Y data
2	Select colour
3	Select character size
4	Select character rotation
5	Line style
6	Swap upper & lower case
7	Reset plotter

File names have no meaning for logical files associated with the plotter.

Disk Drive

Disk drives can be used to store both program and data files. The computer can also send commands and queries to the Disk Operating System (DOS) running in the drive.

An optional drive number followed by a colon may be present before a file name. File names may be up to 16 characters. When addressing an existing file the wildcard characters "*" and "?" may be used.

Opening a Logical File

The disk drive uses the secondary address to distinguish between different types of operation.

SA	*Description*
0	Load file
1	Save file
2-14	Data file
15	Command channel

A file name may be followed by a file type and a mode. Each of these are separated with commas (,). To open a new data file for writing the file type and mode of "W" must be used

```
OPEN 2,8,2,"DATA,S,W"
```

All file types, including program (PRG) files, can be opened.

To overwrite an existing file the file name must be prefixed with "@" character, because of bugs in the 1541 drive it is advised to first delete the file with the SCRATCH DOS command instead.

A data file may be appended to by using a mode of "A". If a file was not closed properly (a splat file) it may be recovered by using a mode of "M".

If an attempt is made to open a logical file with a device number that does not match a disk drive a "Device not present" error is reported. Other errors, such as opening a non-existent file for read will not be detected at the time of opening.

The command channel is used to perform DOS commands and to read the drive status.

A file name must be provided when opening a logical file (unless the secondary address is the command channel).

Saving a Program

To overwrite an existing file the file name must be prefixed with "@" character, because of bugs in the 1541 drive it is advised to first delete the file with the SCRATCH DOS command instead.

A file name must be provided when saving a program to disk. Files will be created with the "P" (PRG) file type.

Secondary addresses have no meaning when saving a program to disk.

Loading a Program

Using a secondary address of 1 causes the program to be loaded at the address it was saved at. Otherwise it is loaded to the start of BASIC (BASIC LOAD command) or the address provided (KERNAL LOAD routine).

A file name must be provided when loading a program from disk. The file type must be "P" (PRG).

Reading the Directory

If the file name used to open a logical file or load a program starts with "`$`" the directory of the disk will be read.

If the secondary address is 0 then the directory contents are returned as a sequence of BASIC lines (link address, line number, data, NUL). If the secondary address is 2 or higher the directory contents are returned as the raw data contained in the BAM and directory blocks on the disk.

Checking Operation Status

The status byte (`ST` variable or result of `READST` routine) has the following bits set after an operation.

Bit	*Description*
0	Write timeout
1	Read timeout
6	End of file
7	Device not present

If a non-existent file is opened for read then a read timeout will be reported when the first read operation (BASIC `INPUT#` or `GET#`, or KERNAL `CHRIN` routine) is performed.

If an existing file is opened for write (rather than append) then both a read & write timeout will be reported when at least two bytes are written (BASIC `PRINT#` or KERNAL `CHROUT` routine).