Simon’s Mostly Reliable Guide to the Commodore Tape Format

Table of Contents

Introduction.....1
Terminology.....1
Contents of the Header.....2
  Header Type.....2
  Start Address.....2
  End Address.....2
  File Name.....2
Program Payload.....3
Duplication.....3
Information Structure.....3
  Synchronization Bytes.....4
  Stream Contents.....4
  Check Byte.....4
Byte Encoding.....4
Symbol Encoding.....4
Error Detection and Recovery.....5
Sequential Files.....5
End of Tape Marker.....6
TAP Files.....6
  Header.....6
  Payload.....6

Introduction

This document tries to explain the “why”, “what” and “how” details of the tape format used by all 8-bit Commodore computers. The specifics mentioned below certainly apply to tapes written by the VIC-20, it is possible (or even likely) that other models have subtle differences to what is described here.

There are several “layers” to discuss, for clarity each will be introduced in turn to, ultimately, build up a complete picture. At times there are topics which are related but not essential to the main subject. These will be prefixed with “Diversion”, they can be skipped if preferred.

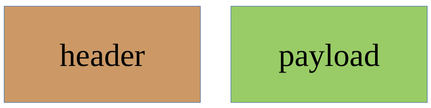
Only the standard Commodore structure and encoding will be discussed, formats used by fastloaders vary and I have not studied them.

Terminology

I will, unashamedly, create my own terms here, these will be shown in *italic* when first mentioned and defined. There are several different ones already in use but they tend to be applied inconsistently, which causes confusion when trying to describe concepts which are similar to one another.

To start with, let us consider program files (BASIC and machine code). Sequential files will be covered once we have dealt with the basics.

Saving a program results in a *header* (containing information about the file) followed by the *payload* (which actually contains the data) being written to tape. Logically a program looks like this:



Contents of the Header

The purpose of the header is to describe what follows it along with properties needed to load the payload to its proper place. All headers have the same length, 192 bytes. A program header contains the following information:

- header type
- start address
- end address
- file name

Let us examine each in turn.

Header Type

There are several forms of payload that can follow. For now let us just accept that a program file is identified using a single byte with the value $03.

Start Address

In order to restore the program to the correct location in memory the starting address is stored as a 16-bit value, as usual this is done as two bytes (first the least significant 8 bits, then the most significant 8 bits).

End Address

To know when the end of the program has been reached the address of the byte beyond the final one of the program is stored in the same way as the start address.

File Name

When saving a file a name can be given to identify the file later. Names are optional and do not have to be unique, they can contain any PETSCII character although trailing spaces are ignored.

Diversion – Program Types

We stated above that program files use a header type of $03. Originally the PET used a header type of $01 for all program files but with the introduction of a variable start to RAM in the VIC-20 a dilemma arose. When a BASIC program is loaded it is stored wherever the current start of memory indicates but a machine code program must be loaded to the same address it was saved at.

To resolve this the header type of $01 is now used for BASIC programs and $03 for other files that must not be relocated. The choice of value is determined by the secondary address supplied to the SAVE command, specifying a secondary address of 1 results in a header type of $03.

Diversion – File Name Length

We did not mention anything about the length of file names. The usual system limit is 16 characters but the KERNAL makes no use of the space after the file name so up to 187 bytes can be stored. This is commonly used by copy protection systems and fast loaders, short machine code routines can be placed here.

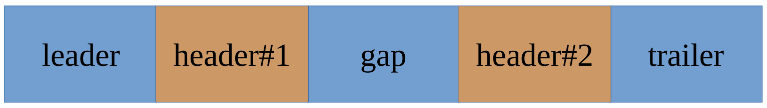
Program Payload

Irrespective of whether the program is BASIC or machine code the payload is made up of a single sequence of bytes. The length of the payload is determined by the difference between the start and end addresses, the payload can contain any 8-bit value.

Duplication

In the diagram above the header and payload were shown as a single block. In reality two copies of each are made. Consumer grade cassette tapes can have physical flaws that affect the reliability of storing data. They also stretch over time. By storing multiple copies data can still be recovered if a small number of errors are detected.

Each copy is preceded and followed by a region that is not data, we will call the region before the first copy the *leader*, the one between copies the *gap*, and the final one the *trailer*.



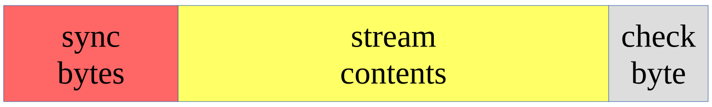
Diversion – Before the Header

At the point a header is to be written the cassette motor is off. When the user presses the Play & Record buttons power is applied and the motor starts running. To allow the motor to reach a constant speed and for the tape tension to become even a period of silence is written before the header.

Information Structure

To take the next step in describing the format we need to generalize, rather than discussing headers and payloads we will talk about *data streams*. All that follows applies to both copies of both headers and payloads.

A data stream consists of a number of *synchronization bytes*, the *stream contents*, and a *check byte*.



Synchronization Bytes

The sync bytes provide a countdown to the stream contents, they consist of the values

$09,$08,$07,$06,$05,$04,$03,$02,$01

In addition they can be used to differentiate between the first and second copy, the first copy also has the top bit set (so the values are actually $89,...,$81).

Stream Contents

The stream contents are a sequence of bytes, depending on the context. For a header this will be the header type followed by the start address etc., for a payload it will be the actual program contents.

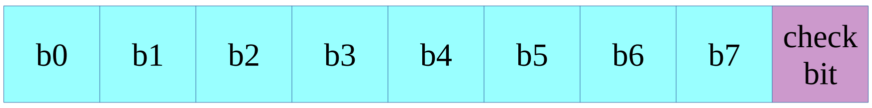
Check Byte

The check byte is an 8-bit checksum computed by XORing the stream contents. It is used to detect corruptions in the stream contents.

Note: the synchronization bytes are not included in the calculation.

Byte Encoding

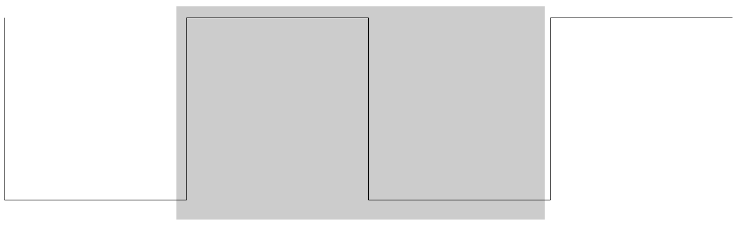
Up until now information has been described in the form of 8-bit bytes. Each byte is actually stored as a sequence of bits, least significant bit first, followed by a *check bit*.



The check bit is an odd parity bit, it is computed by XORing each data bit.

Symbol Encoding

Digital computers store data in memory as binary values, storing information on tape is done as pulses of opposing magnetic patterns. These are seen by the computer hardware as a sequence of *cycles*


One cycle

Rather than encoding binary values as single cycles the Commodore format uses *cycle pairs* of three different lengths. For simplicity we will call these symbols *short*, *medium* and *long*.

Bit value	*Symbol pair*
0	**S**, **M**
1	**M**, **S**

Each byte is introduced by the symbol pair **L**, **M** indicating *start of data*. This allows the byte boundary to be recovered even if one or more data bits become corrupted. In total 20 cycles are used to encode a single byte (2 for start of data, 16 for 8 bits of data, 2 for the check bit).

Between each data stream the leader, gap and trailer are encoded as a sequence of short cycles

	Cycle count
Leader	First $6A00, second $1A00
Gap	$50
Trailer	$50

At the end of each data stream a long cycle is appended. Combined with the following gap (or trailer) this means the sequence **L**, **S** indicates the end of a data stream.

Error Detection and Recovery

As described above each data stream is duplicated and each byte within it has a check bit. When reading the first copy of a data stream the location of any bytes that have a check bit mismatch are recorded in an error log. As the second copy is read if the check bit is correct the value is stored. Space in the error log is limited, only 30 errors may be corrected in this way.

When the second copy has been completely read the checksum is computed and compared with the check byte at the end of the data stream. If there is a mismatch a checksum error is stored in the ST variable.

Sequential Files

In addition to program files tapes can also be used to store a variable amount of data. Sequential files are created by the OPEN command rather than the SAVE command. This writes a header with a header type of $04, the start and end addresses have no meaning for a sequential file and the space beyond the file name is unused.

As the final amount of data in the file is unknown it is written and read in blocks. When writing data to a sequential file each byte is placed into the tape buffer until it contains 191 bytes. The first byte of the buffer contains the value $02 to indicate this is a data block. Once full the entire tape buffer is written to tap. When the file is closed a final $00 byte is added to the file and a full data block is written.

When reading a sequential file the KERNAL reads each data block into the tape buffer in the same way as it does a header. The end of the file is indicated by the first $00 byte, this value cannot be returned by the GET# command.

End of Tape Marker

When searching for a file by name on a tape a match may not be found. To terminate the search on the last file on the tape an *end of tape* marker can be written. By specifying a secondary address of 2 a final header with a header type of $05 is written after the payload.

If a header with type $05 is read while searching for a file the operation terminates with a ?DEVICE NOT PRESENT error rather than setting the ST variable. This seems to be accidental, the value $05 is just used in the usual error code path rather than being converted to a status bit.

TAP Files

Many emulators support loading tape images from a .TAP file. These files consist of a fixed header followed by a stream of cycles, the format is flexible and can contain encodings other than the standard Commodore format.

Header

The header consists of 20 bytes at the start of the file

Field	*Description*
Signature	The string “C64-TAPE-RAW”
Version	0 or 1
Platform	0=C64, 1=VIC-20, 2=C16
Video	0=PAL, 1=NTSC1, 2=NTSC2
Reserved	Always 0
Data size	Size of data payload that follows, 32-bit value

A variant format with a signature of “C16-TAPE-RAW” and a version of 2 exists.

Payload

The data that follows the header represents the time period between cycles, specifically the falling edge as it transitions from high to low. This is the event the KERNAL acts on via interrupts.

Non-zero values are the number of CPU clock cycles since the last transition divided by 8.

The value zero has a different meaning depending on file version. In a version 0 file they represent a period in excess of 2040 CPU clock cycles. In a version 1 file the period is encoded in the following three bytes, least significant byte first.