Tapecart Programmer's Reference

This document describes the Tapecart from a programmer's perspective - i.e., how to communicate with it to read and write data.

Intro

The tapecart is a small module connected to the C64's tape port which allows a program running on the C64 to access its on-board flash memory. It supports three different modes:

1. Streaming mode
2. Fastload mode
3. Command mode

Modes are described in detail below. The tapecart always starts in streaming mode to allow the user to load an initial program using Shift+Run/Stop. To ensure this load operation happens as fast as possible, the tapecart only supports a two-byte-long program, overwriting a vector to cause an auto-start in the tape buffer. This auto-started code can be programmed independently of the main flash, but a default version is supplied that uses fastload mode to load a pre-defined block of flash into the C64's memory and starts it.

Switching from streaming mode to either fastload mode or command mode is accomplished by sending a bit sequence using the write and motor lines, which is detailed in the "Streaming mode" section below.

To ensure that the user can always run the initial program again, the tapecart needs to detect if the C64 is reset and return to streaming mode. It does this by monitoring the motor line: If at any point in the Fastload or Command modes the motor line becomes active, it immediately returns to streaming mode. To switch back to the previous mode, the bit sequence needs to be sent again.

The fastload mode is intended to be mainly used by the default loader. It sends a small header to the C64 followed by data read from the flash memory. It uses a fast 2-bit protocol that can reach transmission speeds of around 9500 bytes/second. If only a single one-filed program is intended to be loaded from the tapecart, this mode is all that is needed and the flashtool can set up everything as needed while writing the program to flash. The default loader uses a JMP to a programmable start address to start the program it has loaded, but it is simple to add a small header that simulates a RUN for programs started from BASIC. More defailts about the fastload mode are given in the section titled "Fastload mode".

The third, and most flexible mode is command mode. In this mode, the tapecart listens for commands that can be used to read, wrrite and erase the flash memory as well as set parameters like the length of the data block used for fastload mode. With the exception of the READ_FLASH_FAST command, all data is transmitted and received using 1-bit protocols that are not timing critical - so they could for example be used to load data while raster interrupts are still used for effects. Of course a variation of the fast 2-bit protocol used by the fastload mode is also available, but it cannot tolerate interrupts, badlines or sprites.

Since the tapecart is based on flash memory, it can be read freely from any starting byte, but writing is more limited. Data is organized in pages of a fixed length and only full pages should be written - writing a partial page results in undefined data in the remaining bytes of the page. Furthermore, before a page can be written, it must be erased which is only possible in blocks that are a multiple of the page size and aligned to the same multiple. The exact values for these parameters may vary between tapecarts depending on the availability of cheap flash chips, so a command is available to read the details of the tapecart that is currently connected. It is expected though that all carts will stay within these parameters:

- memory size 2 MByte (2,097,152 byte) or more, although a 64KByte version for onefilers might appear
- page size 4096 bytes or less (currently 512 byte on early prototypes, 256 byte on initial release)
- erase block size 64 KByte or less (64KB on early prototypes, 4KB on initial release)

To simplify working with tapecarts that have different erase block sizes, a command is available that will always erase a 64 KByte block (aligned at a 64 KByte boundary), no matter what the actual erase block size is. If you use the tapecart as a read-only device, you do not need to worry about page size or erase block sizes since the flashtool will handle these details for you. However, if you plan to write to the tapecart yourself from your application, you need to plan your memory layout to ensure that no vital data is erased when you erase a block before writing to it.

Streaming mode

In streaming mode, the tapecart basically behaves like a datasette with an endless tape. It sends a pulse stream to the C64 that contains an auto-starting initial loader program in the tape buffer, which uses $0302 as its auto-start vector. The default initial loader switches to fastload mode to load the main program and tries to start it using RUN. A custom initial loader can be installed using the flash tool (or the WRITE_LOADER command), but it is limited to 171 bytes and will always be started at $0351 using the $0302 vector.

After each transmission of the initial loader, the tapecart releases the sense line of the tape port for about 200 milliseconds. This is necessary to resume loading in case an initial load attempt was cancelled using RUN/STOP. After this delay has passed, but before restarting the pulse stream, the tapecart checks if it has received a mode switch sequence. The tapecart is switched to fastload or command mode by sending one of two bit sequences on the write line. Each time the motor is turned on, the state of the write line is sampled and a 16-bit shift register is shifted left by one bit, using the current value on the write line as its lowest bit. A rough approximation as C code:

```
shift_reg = shift_reg << 1;
if (get_write())
  shift_reg = shift_reg | 1;
```

The tapecart checks the contents of the shift register occasionally, the maximum time between two checks should be less than two milliseconds. If the shift register contains the value $ca65, it enters fastload mode; if the value is $fce2, it enters command mode. The shift register is reset every time the transmission of the initial loader's header block, so it may be a good idea to start transmitting it only if the sense line is low.

Please note that turning off the motor is very slow on some C64 board revisions, it can take about a millisecond (~1000 cycles) until the signal has decayed sufficiently. On the other hand, turning on the motor is very fast and no delay should be needed - do make sure that the level on the write line has already been set before turning the motor bit in $01 on, otherwise the mode switch sequence may not be detected reliably. You should turn off the motor after the sequence has been transmitted, otherwise the tapecart will immediately return to streaming mode.

A sample implementation for the mode switch sequence with a value of $ca65 can be found in the initial loader, a sample implementation for entering command mode can be found in the C sources of the flash tool (tapecartif.c).

Fastload mode

Fastload mode transmits a selected part of the flash memory using a fast 2-bit fixed-timing protocol which reaches ~9500 bytes/second. The protocol is similar (but not identical) to the fast 2-bit read used in command mode - you shouldn't need to worry about it unless you want to make extensive modifications to the default initial loader.

Fastload mode can transmit up to 65535 bytes to the C64 which should be sufficient for most purposes. The length and offset of the data block in flash memory as well as the start address that the initial loader should jump to after loading can be set using the WRITE_LOADINFO command. The default initial loader can load data anywhere in the C64's RAM, even in the RAM below I/O at $d000. It treats the first six bytes it receives as an "info block" which specifies the start and end addresses of the data in the C64 as well as the address to jump to after loading. You do not need to worry about this info block though, the tapecart automatically calculates it for you.

In the flash memory, the program to be loaded is structured exactly the same as a PRG file: Two bytes that specify the load address, followed by data. Since there is no way to detect the length of a PRG file, the number of bytes (including the two bytes for the load address) must be set using the WRITE_LOADINFO command. Please be aware that it may be a bad idea to load over the tape buffer because the initial loader runs from there, or over zeropage addresses $aa to $af where the default initial loader stores a few bytes of data.

The default initial loader uses a JMP to start the program it has loaded. At this point, the autostart vector has been reset to its default value, the screen is turned back on and interrupts are enabled again. If you need to start a BASIC program with RUN instead, you can add a short header just before $0801 that does a few calls into the BASIC ROM to simulate RUN - the flashtool contains an implementation of this for its onefiler-writing mode. The contents of the CPU registers are unspecified and may change without notice, except as described in the section *Data Block Offset option* below.

Command mode

In command mode, the tapecart expects to receive commands using a 1-bit C64-timed protocol (similar to an IRQ loader for the 1541) and sends its answer with a 1-bit C64-timed protocol unless the command specifies something different.

When the tapecart has detected the command mode magic value at the end of the 200ms pause, it sets the sense line high and waits until the write line is high. Even though the motor line is off at this time, it will resume sending pulses until the write line is low again, after which it waits to receive a command. If at any time during this process the motor line turns on again, the tapecart aborts and enters streaming mode again. In tabular form, the process looks like this:

1. send the magic number for command mode ($fce2) - the motor should be off after sending it
2. delay for one millisecond (~1000 cycles) to ensure that the motor signal is really off (unless the delay is already at the end of your send loop in step 1)
3. set the write line high

4. wait until the sense line is high
5. wait until at least three pulses on the read line have been received (CIA 1 ICR $dc0d, bit 4)
6. set write low

To send a command to the tapecart in this mode, send a single byte with the command code to the tapecart using the byte transmit scheme described below, followed by any parameter bytes that may be needed (zero for some commands). If the command returns data, read it using the byte receive scheme described below or the fast 2-bit read for the READ_EXTMEM_FAST command. Currently, sample implementations of the transmit and receive schemes only exist in C form in the sources of the flashtool (tapecartif.c).

While not sending a command, the write line should be low and the sense line should be configured as input. The tapecart sets sense low while it is busy and high to signal that it is ready to receive a byte

> this was chosen so that the kernal IRQ handler can be told to not turn on the motor (which would exit command mode) by writing a non-zero value in $c0.

Command mode stays active after a command is processed. It will be exited only if an unknown command byte is received, the EXIT command is received or the motor line becomes active (used to detect a C64 reset). With the current code base, the device's LED is on while it is in command mode and turns off as soon as it is exited, but there are also two commands to explicitly turn the LED on or off.

Byte transmit (C64->Tapecart)

If you use the kernal interrupt handler, it is recommended to disable interrupts while sending bytes to the tapecart because the kernal's interrupt handler would turn on the motor, kicking the tapecart out of command mode. It can also be useful to set $c0 to a non-zero value after transmitting a byte but before re-enabling interrupts because the tapecart may signal a busy state with the sense line low for some time. A non-zero value in $c0 ensures that the kernal's interrupt handler will not turn on the tape motor in this case.

The byte transmission is a C64-clocked protocol, using the write line as a clock signal and the sense line to transmit data. Please note that in this case the direction of the sense line is opposite to the normal usage in the C64, so bit 4 of $00 must be set to 1 temporarily while a byte is sent. In tabular form, the protocol looks like this:

1. If you use the kernal interrupt handler, turn off interrupts
2. wait until sense is high (tapecart is ready to receive)

3. switch the sense line to output (set bit 4 of $00)
4. for all 8 bits of the byte, repeat the following, starting with the most-significant bit:
    1. set the sense line to the value of the bit
    2. set the write line high
    3. set the write line low
5. set the sense line high
6. set the sense line low
7. set the sense line to input again (clear bit 4 of $00)
8. If you use the kernal interrupt handler, set $c0 to a non-zero value and re-enable interrupts

Please note that steps 5-7 are needed so the tapecart can set its own sense pin to output without risking that the C64's sense pin is also still set to be an output. The tapecart will switch its sense pin to output 10 microseconds (~10 cycles) after it sees the high->low transition and since the state of the C64's pin is known at this point, this is safe even if step 7 on the C64 is delayed, e.g. due to a badline.

After receiving a byte, the tapecart will set the sense line low until it has finished processing the received byte.

Byte receive (Tapecart->C64)

If you use the kernal interrupt handler, it is recommended to disable interrupts while receiving bytes from the tapecart because the kernal's interrupt handler would turn on the motor, kicking the tapecart out of command mode. It can also be useful to set $c0 to a non-zero value after receiving a byte but before re-enabling interrupts because the tapecart may signal a busy state with the sense line low for some time. A non-zero value in $c0 ensures that the kernal's interrupt handler will not turn on the tape motor in this case.

The byte reception is a C64-clocked protocol, using the write line as a clock signal and the sense line to transmit data. Unlike the transmit protocol, all lines are used in their standard direction, so no accesses to $00 are needed. In tabular form, the protocol looks like this:

1. If you use the kernal interrupt handler, turn off interrupts
2. wait until sense is high (tapecart is ready to receive)
3. set the write line high
4. repeat 8 times for the bits of a byte, the most-significant bit is received first:
    1. set the write line low
    2. set the write line high
    3. if the sense line is high, a 1 bit has been received; if it is low, a 0 bit has been received

5. set the write line low
6. If you use the kernal interrupt handler, set $c0 to a non-zero value and re-enable interrupts

After sending a byte, the tapecart will set the sense line low until it is ready to transmit (or receive) another byte.

Commands

A number of commands are defined for the tapecart. Some of them are mostly useful for the flashing tool, but have been documented here anyway for completeness. If the tapecart receives a command byte that it does not know, it switches to streaming mode.

Some commands expect to receive additional bytes with command arguments. Some commands send a reply, some do not. Numerical values that exceed 8 bit are sent in little endian (low byte first). Both parameters and replies are given in the order they are expected/sent.

Command $00: EXIT

This command exits to streaming mode.

No parameters, no reply.

Command $01: READ_DEVICEINFO

This commands reads a string containing a device identification.

No parameters.

Reply:

PETSCII string, 0-terminated

Command $02: READ_DEVICESIZES

This command reads the total size of the tapecart's memory, its page size and its erase block size.

No parameters.

Reply:

1. 3 byte total size (in byte)
2. 2 byte page size (in byte)

3. 2 byte erase block size (in pages!)

If the erase block size is 0, direct byte write is supported and no erase commands are needed.

Command $03: READ_CAPABILITIES

This command reads a bit field that specifies which extended capabilities are available on this tapecart. If an additional capability is available, its bit is set to 1. Currently, no extended capabilities are defined, so the value returned by this command consists of 4 $00 bytes.

No parameters.

Reply:

1. 4 byte flags

Command $10: READ_FLASH

This commands reads from the tapecart's flash memory.

Parameters:

1. 3 byte start address
2. 2 byte length (in bytes)

Reply:

1. n byte data

Attempting to read using a length of 0 bytes results in undefined behaviour. Reading from an address beyond the flash's size (either by specifying an invalid start address or a length that results in an access beyond the flash size) results in undefined behaviour.

Command $11: READ_FLASH_FAST

This command reads from the tapecart's flash memory and sends the data with a fast 2-bit protocol.

Parameters:

1. 3 byte start address
2. 2 byte length (in bytes)

Reply:

This command does not use the usual 1-bit protocol to reply!

1. n byte data

Attempting to read using a length of 0 bytes results in undefined behaviour. Reading from an address beyond the flash's size (either by specifying an invalid start address or a length that results in an access beyond the flash size) results in undefined behaviour.

Since the alternative protocol used by this command is timing sensitive, below is a sample code snippet that implements a compatible byte reception subroutine. If you modify it, please ensure that all read/write accesses and direction changes on the write/sense lines happen at the same clock cycle as the original, relative to the point where the write line is set high. The timing was chosen to be safe on both PAL and NTSC machines, even if the internal clock of the tapecart's controller has worst-case deviation.

The code assumes that the CPU will not be interrupted by any VIC memory access. The easiest way to ensure this is to blank the screen and wait for 20ms, but any method that ensures that the cycle-counted part of the code is not delayed should work.

```
;;   getbyte_fast.s: Fast byte read
;;
;;   Note: This uses the opposite level on sense to determine ready-ness
;;         compared to the loader mode!
;;

        .export _tapecart_getbyte_fast
_tapecart_getbyte_fast:
        ;; wait until tapecart is ready (sense high)
        lda #$10
rdyloop:
        bit $01
        beq rdyloop

        ;; (this would be a nice place to check if a badline is coming up)

        ;; send our own ready signal
        ldx #$38
        lda #$27
        stx $01         ; set write high (start signal)
        sta $00         ; 3 - switch write to input
        nop             ; 2 - delay

        ;; receive byte
        lda $01         ; 3 - read bits 5+4
        and #$18        ; 2 - mask
        lsr             ; 2 - shift down
        lsr             ; 2
        eor $01                 ; 3 - read bits 7+6
        lsr                     ; 2
        and #$0f                ; 2 - mask
        tax                     ; 2 - remember value

        lda $01                 ; 3 - read bits 1+0
        and #$18                ; 2 - mask
        lsr                     ; 2 - shift down
        lsr                     ; 2
        eor $01                 ; 3 - read bits 3+2
        lsr                     ; 2
        and #$0f                ; 2 - mask
        ora nibbletab,x         ; 4 - add upper nibble

        ldx #$2f                ; 2 - switch write to output
        stx $00                 ; 3
        ldx #$36                ; set write low again
        stx $01

        rts

nibbletab:
        .byt $00, $10, $20, $30, $40, $50, $60, $70
        .byt $80, $90, $a0, $b0, $c0, $d0, $e0, $f0
```

Please note that the protocol used by this command is similar, but not identical to the one used in fastload mode.

Command $12: WRITE_FLASH

This command writes to the tapecart's flash memory.

Parameters:

1. 3 byte start address
2. 2 byte length (in bytes)
3. n byte data

No reply

Attempting to write using a length of 0 bytes results in undefined behaviour. Writing to an address beyond the flash's size (either by specifying an invalid start address or a length that results in an access beyond the flash size) results in undefined behaviour.

The tapecart's firmware will ensure that page-crossings are handled correctly if you want to write more than one page, but it does not auto-erase the memory that is written to. If the data does not end at the end of a page, the new content of the remaining bytes of the page is undefined.

Command $13: WRITE_FLASH_FAST

This command is currently not implemented.

Command $14: ERASE_FLASH_64K

This command erases a 64KB block in the flash memory.

Parameters:

1. 3 byte address

The address can specify any byte within the intended 64KB block.

Command $15: ERASE_FLASH_BLOCK

This command erases a single erase block in the flash memory.

Parameters:

1. 3 byte address

The address can specify any bytes within the intended erase block. The size of the erase block can be determined by using the READ_DEVICESIZES command.

Command $16: CRC32_FLASH

This command calculates the CRC32 of an area of the flash memory contents.

Parameters:

1. 3 byte start address
2. 3 byte length (in bytes)

Reply:

1. 4 bytes CRC32

This command uses the standard CRC32 algorithm (polynomial 0x04c11db7), the same as e.g. Ethernet, ZIP or PNG.

Command $20: READ_LOADER

This command reads the current initial loader.

No parameters

Reply:

1. 171 byte loader

Command $21: READ_LOADINFO

This command reads the current initial loader auxillary data.

No parameters

Reply:

1. 2 byte data address
2. 2 byte data length
3. 2 byte call address
4. 16 byte file name

The data address and length in the reply refer to the tapecart's external memory - the fastload mode starts to read at this address and transmits as many bytes as specified here. After loading, it jumps to the call address returned by this command. The file name is the name shown by the C64 in the "FOUND ..." message when the user loads the initial loader.

Command $22: WRITE_LOADER

This command writes a new initial loader to the tapecart.

Parameters:

1. 171 byte loader

No reply

Command $23: WRITE_LOADINFO

This command updates the initial loader auxilary data.

Parameters:

1. 2 byte data address
2. 2 byte data length
3. 2 byte call address
4. 16 byte file name

No reply

The data address and length refer to the tapecart's external memory - the fastload mode starts to read at this address and transmits as many bytes as specified here. After loading, the default initial loader jumps to the call address set by this command. The file name is the name shown by the C64 in the "FOUND ..." message when the user loads the initial loader.

Command $30: LED_OFF

This command turns the LED on the tapecart off (if available). Please note that the LED is turned on when command mode is entered.

No parameters, no reply.

Command $31: LED_ON

This command turns the LED on the tapecart on (if available). Please note that the LED is already turned on when command mode is entered.

No parameters, no reply.

Command $32: READ_DEBUGFLAGS

This command reads the current debug flags. See WRITE_DEBUGFLAGS below for details about the flags.

No parameters

Reply:

1. 2 byte debug flags

Command $33: WRITE_DEBUGFLAGS

This command sets the debug flags. These flags are intended to enable features that may be useful when writing or debugging programs that access the tapecart. The flags are non-persistent, a power cycle will set all of them to 0 again. To enable a debug flag, set its bit in the parameter of this command to 1; to disable it, set it to 0. To avoid any chicken-and-egg problems when sending this command, the flash tool has an option to set the debug flags.

Parameters:

1. 2 byte debug flags

No reply

Currently, there are two defined debug flags:

1. SEND_CMDOK ($0001) - sends the characters 'O' and 'K' before a command byte is read. Probably not that useful when accessing a tapecart via the tape port, but helpful in the undocumented UART mode. You cannot set this flag with the menu option in the flash tool.
2. BLINK_MAGIC ($0002) - show the current content of the magic value shift register by blinking the LED after each kernal loader cycle. The value is shown MSB-first, a long pulse of the LED stands for a 1-bit and a short pulse for a 0-bit. Please note that not all tapecarts are equipped with a LED and even on those they are the LED may not work.
3. BLINK_COMMAND ($0004) - similar to BLINK_MAGIC, but shows the received command byte by blinking

Command $40: DIR_SETPARAMS

This commands sets the parameters for the directory lookup command (see below). To simplify porting file-based programs to the tapecart, the cart can look up a given "file name" in a directory stored in the flash memory and return fixed-length data (e.g. a flash address and length) associated with that file name.

Parameters:

1. 3 bytes flash address of the start of the directory
2. 2 bytes number of entries in the directory
3. 1 byte length of a file name in the directory (values over 16 will be limited to 16), called "n" below
4. 1 byte length of the data associated with a file name, called "m" below

No reply

Please see below for a description of the directory format.

Command $41: DIR_LOOKUP

This command looks up a file name in the directory that was set using DIR_SETPARAMS and returns the associated data bytes if the name was found.

Parameters:

1. n bytes file name (n set by the third parameter of DIR_SETPARAMS)

Reply:

1. 1 byte "found" marker: 0 if the name was found, nonzero if not

2. (only if byte 1 was 0) m byte data (m set by the fourth parameter of DIR_SETPARAMS)

A directory consists of a number of entries, each of which is a fixed-length record with a length of n+m bytes. Each record contains a file name (n bytes) followed by the data associated (m bytes) with that file name. You get to decide yourself how that data is structured, for example you could use the flash-address and length of the actual file contents. The length of the data part of each entry in the directory must be the same! File name comparison uses binary matching with no wildcards, every single byte of the name to be looked up must match the name in the flash exactly.

Commands from $f0 on are reserved for system functionality like firmware updates and will not be documented here.

Data Block Offset option

In order to allow multiple tapecart-utilizing programs to coexist on the same tapecart, programs may support data block offsets. Such a program is able to run even if its data is not stored starting from flash address 0 by adding an offset to all addresses it sends to the tapecart.

The selection of the program to start would be made by another application which is started by the tapecart's initial loader, for example a menu system allowing the user to choose from a list of programs available on the tapecart. Implementation details for such a menu system are beyond the scope of this reference, except for the method used to pass the correct data block offset to the program to be started.

A program that wants to declare support for data block offsets must be supplied in TCRT format and it must set bit 1 of the misc. flags field (see the [TCRT format specification](TCRT Format.md) for details). Since a user may write such a program directly to a tapecart without the use of an additional menu program, it must be able to run even if no data block offset has been passed to it. This can easily be implemented by using a default data block offset of 0 and replacing that with the actual offset if present.

Data block offsets are always integer multiples of the erase block size, i.e. a multiple of 4096 byte for non-prototype tapecarts. In the unlikely event that a future tapecart supports erase block sizes that are not divisible by 256, the data block offset will be chosen so that it is a multiple of 256.

Checking for a data block offset

To indicate that a data block offset should be used, the loader (e.g. the initial loader or a menu application) sets the Y register to $4f (lower-case "o" in PETSCII). The data block offset itself is stored in the zero-page at addresses $00fb and $00fc. If the Y register contains any other value, no data block offset has been passed and it should be assumed that the offset is 0.

An application that does not support data block offsets or that does not need to read or write data from the tapecart at all may ignore any data block offset passed to it.

Address $00fb in the zero-page holds the middle 8 bits of the data block offset and address $00fc holds the top 8 bits of the data block offset. The lowest 8 bits are not explicitly given anywhere, but can be assumed to be 0, because the offset will always be a multiple of 256 or more.

Precautions when working with data block offsets

You must make sure that you add the data block offset to all flash addresses passed to the tapecart, both for read and write accesses. How you do this is up to you - for example you could use self-modifying code to change all addresses in your code or you could add it before every access.

Do make sure that you correctly handle the offset for write accesses - reading from an incorrect address will just give wrong data to your program, but writing to an incorrect address will corrupt other programs on the tapecart.

For additional safety, you may want to use the CRC32_FLASH command to verify that the flash content starting at the current data block offset is actually the data you expect to be there. This could also be used to detect corruption caused by other programs writing to the wrong flash address. If you do verify your data this way, remember to exclude all flash pages that your program writes into.