Table of Contents

[Main Loop 2](#_Toc451613985)

[ILDA 2](#_Toc451613986)

[get\_command 2](#_Toc451613987)

[Stream Format Parser 2](#_Toc451613988)

[Marlin Code 3](#_Toc451613989)

[get\_command 3](#_Toc451613990)

# Main Loop

## ILDA

### get\_command

If the buffer has room for the next line then read the next character.

The main loop uses the function named get\_command to retrieve characters from the serial port.

The function named get\_command contains a while loop in which the serial is queried via a call to MSerial.available().

Each character from the serial port is saved to the buffer named cmdbuffer. This buffer is a 2 dimensional array: 4(BUFSIZE) x 96(MAX\_CMD\_SIZE).

The cmdbuffer is an array of strings! There are a maximum of 4 strings that can be processed at a time. The Marlin firmware will retrieve 4 strings from the serial stream.

# Stream Format Parser

When the stream begins, look for the text ILDA. This text is the beginning of an ILDA header section. If this is not detected then start from the beginning and look for the text again.

For now, the format code and the number of records count will be retrieved. All of the other header information is not needed at this time.

The format code defines the structure of the data record. For our purposes, format code 1 should suffice. Format code 1 is defined as having 2D coordinates with a status code and the indexed color.

## Marlin Code

### get\_command

* while there is still data available from the serial port AND the variable buflen is less than the buffer’s string count
  + read the next character from the serial port
  + if the character is a ‘\n’ OR ‘\r’ OR (‘:’ AND comment\_mode is false) OR the current buffer’s string array length is exceeded then
    - if empty line in buffer(serial\_count is zero)
      * set comment\_mode to false
      * return from get\_command
    - terminate the buffer’s string
    - if !comment\_mode
      * set comment\_mode to false
      * set current SD buffer to false
      * search the buffer’s string for the ‘N’ character
      * if ‘N’ is found
        + set strchr\_pointer to the location of the ‘N’ character
        + extract the numeric value after the ‘N’ character and set gcode\_N with the value
        + if the current ‘N’ value is not equal to the last ‘N’ value plus 1 AND the text ‘M110’ is not found in the buffer’s string

set the error

request a flush and resend

set the buffer’s string index to the beginning

return from get\_command

* + - * + search the buffer’s string for the ‘\*’ character

calculate the checksum by starting at the beginning of the buffer’s string and iterating through each character until the ‘\*’ character is found

set strchr\_pointer to the location of the ‘\*’ character

extract the double value starting from the ‘\*’ character to the end of the buffer’s string then convert to an integer. This is the checksum.

If the calculated checksum is not equal to the checksum read

set the error

request a flush and resend

set the buffer’s string index to the beginning

return from get\_command

* + - * + else

set the error

request a flush and resend

set the buffer’s string index to the beginning

return from get\_command

* + - * + set the gcode\_LastN equal to gcode\_N
      * else
        + search the buffer’s string for the ‘\*’ character
        + if the ‘\*’ character is found

set the error

request a flush and resend

set the buffer’s string index to the beginning

return from get\_command

* + - * search the buffer’s string for the ‘G’ character
      * if the ‘G’ character is found
        + set strchr\_pointer to the location of the ‘G’ character
        + extract the int value starting from the ‘G’ character to the end
        + if the value is 0, 1, 2, or 3

if the variable Stopped is true

present ‘stopped’ message to LCD screen

* + - * if the buffer’s string is equal to the text ‘M112’
        + invoke kill()
      * increment the buffer’s string index and then modulo it will 4(the number of strings the buffer contains)
      * increment the variable buflen
    - clear the buffer’s string by setting the index serial\_count to zero
  + else
    - if the character is a ‘;’
      * set comment\_mode to true
    - if !comment\_mode
      * add the character to the end of the buffer’s string and increment the string index
      * code: cmdbuffer[bufindw][serial\_count++] = serial\_char