Table of Contents

[Variables 2](#_Toc451713187)

[ILDA Parsing 3](#_Toc451713188)

[Header Parser 3](#_Toc451713189)

[Header 3](#_Toc451713190)

[Bytes 1 – 4 3](#_Toc451713191)

[Bytes 5 – 7 3](#_Toc451713192)

[Bytes 8 3](#_Toc451713193)

[Bytes 9 – 16 4](#_Toc451713194)

[Bytes 17 – 24 4](#_Toc451713195)

[Bytes 25 – 26 4](#_Toc451713196)

[Bytes 27 – 28 4](#_Toc451713197)

[Bytes 29 – 30 4](#_Toc451713198)

[Bytes 31 4](#_Toc451713199)

[Bytes 32 4](#_Toc451713200)

[Pseudo code 4](#_Toc451713201)

[Record Parser 5](#_Toc451713202)

[Marlin Code 7](#_Toc451713203)

[get\_command 7](#_Toc451713204)

# Variables

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.

# ILDA Parsing

## Header Parser

### Header

The header has a fixed size of 32 bytes.

#### Bytes 1 – 4

Will contain the characters “ILDA”.

#### Bytes 5 – 7

Reserved

#### Bytes 8

Contains one of the following format codes:

* Format 0 – 3D coordinates with indexed color
  + Record size is 8 bytes
    - Bytes 1 – 2 are the X coordinates
    - Bytes 3 – 4 are the Y coordinates
    - Bytes 5 – 6 are the Z coordinates
    - Byte 7 is the status code
    - Byte 8 is the color index
* Format 1 – 2D coordinates with indexed color
  + Record size is 6 bytes
    - Bytes 1 – 2 are the X coordinates
    - Bytes 3 – 4 are the Y coordinates
    - Byte 5 is the status code
    - Byte 6 is the color index
* Format 2 – color palette for indexed color frames
  + Record size is 3 bytes
    - Byte 1 is the red color value
    - Byte 2 is the green color value
    - Byte 3 is the blue color value
* Format 4 – 3D coordinates with true color
  + Record size is 10 bytes
    - Bytes 1 – 2 are the X coordinates
    - Bytes 3 – 4 are the Y coordinates
    - Bytes 5 – 6 are the Z coordinates
    - Byte 7 is the status code
    - Byte 8 is the blue color value
    - Byte 9 is the green color value
    - Byte 10 is the red color value
* Format 5 – 2D coordinates with true color
  + Record size is 8 bytes
    - Bytes 1 – 2 are the X coordinates
    - Bytes 3 – 4 are the Y coordinates
    - Byte 5 is the status code
    - Byte 6 is the blue color value
    - Byte 7 is the green color value
    - Byte 8 is the red color value

#### Bytes 9 – 16

Container for the frame or color palette name.

#### Bytes 17 – 24

Contains the company name.

#### Bytes 25 – 26

Holds the number of records in big endian format.

#### Bytes 27 – 28

Hold the frame or color palette number.

#### Bytes 29 – 30

Total frames or zero.

#### Bytes 31

Project number.

#### Bytes 32

Reserved

### Pseudo code

When the stream begins, look for the text “ILDA”. 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.

* Read a character
* Is it an ‘I’
  + No, start from the beginning
  + Yes, then get the next character
* Read a character
* Is it an ‘L’
  + No, start from the beginning
  + Yes, then get the next character
* Read a character
* Is it an ‘D’
  + No, start from the beginning
  + Yes, then get the next character
* Read a character
* Is it an ‘A’
  + No, start from the beginning
  + Yes, then we found the start of the header
* Consume characters until we get the format code
* Record the format code
* Consume characters until we get the number of records
* Record the number of records
* Consume character until we reach the end of the header

## Record Parser

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.

Switch on the format code. The default action on the switch statement will signal an error.

### Format 0

3D coordinates with indexed color. Record size is 8 bytes.

* Bytes 1 – 2 are the X coordinates
* Bytes 3 – 4 are the Y coordinates
* Bytes 5 – 6 are the Z coordinates
* Byte 7 is the status code
* Byte 8 is the color index

#### Pseudo code

* Calculate how many records can fit in 1 buffer’s string of 96 bytes
* Read these bytes into the buffer
* When at the end of the buffer’s string, increment to the next string
* Continue reading bytes until the entire buffer is full then do not read any more characters until a buffer’s string is empty

### Format 1

2D coordinates with indexed color. Record size is 6 bytes

* Bytes 1 – 2 are the X coordinates
* Bytes 3 – 4 are the Y coordinates
* Byte 5 is the status code
* Byte 6 is the color index

#### Pseudo code

* Calculate how many records can fit in 1 buffer’s string of 96 bytes
* Read these bytes into the buffer
* When at the end of the buffer’s string, increment to the next string
* Continue reading bytes until the entire buffer is full then do not read any more characters until a buffer’s string is empty

### Format 2

Color palette for indexed color frames. Record size is 3 bytes.

* Byte 1 is the red color value
* Byte 2 is the green color value
* Byte 3 is the blue color value

#### Pseudo code

* Calculate how many records can fit in 1 buffer’s string of 96 bytes
* Read these bytes into the buffer
* When at the end of the buffer’s string, increment to the next string
* Continue reading bytes until the entire buffer is full then do not read any more characters until a buffer’s string is empty

### Format 4

3D coordinates with true color. Record size is 10 bytes.

* Bytes 1 – 2 are the X coordinates
* Bytes 3 – 4 are the Y coordinates
* Bytes 5 – 6 are the Z coordinates
* Byte 7 is the status code
* Byte 8 is the blue color value
* Byte 9 is the green color value
* Byte 10 is the red color value

#### Pseudo code

* Calculate how many records can fit in 1 buffer’s string of 96 bytes
* Read these bytes into the buffer
* When at the end of the buffer’s string, increment to the next string
* Continue reading bytes until the entire buffer is full then do not read any more characters until a buffer’s string is empty

### Format 5

2D coordinates with true color. Record size is 8 bytes.

* Bytes 1 – 2 are the X coordinates
* Bytes 3 – 4 are the Y coordinates
* Byte 5 is the status code
* Byte 6 is the blue color value
* Byte 7 is the green color value
* Byte 8 is the red color value

#### Pseudo code

* Calculate how many records can fit in 1 buffer’s string of 96 bytes
* Read these bytes into the buffer
* When at the end of the buffer’s string, increment to the next string
* Continue reading bytes until the entire buffer is full then do not read any more characters until a buffer’s string is empty

# Marlin Code

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().

## 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 with the NULL character
    - 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
    - set the buffer’s string index named 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