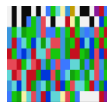


# Khroma Coder



Updated to 1.0.18 (20/07/2023)  
By Karst Skarn (Discord *owain.doomer*)





## Disclaimer

*Given the multi-purpose nature of this program, I must declare that I am not responsible for any use that may be made of it, nor for any damages that may be incurred by the use of this program.*

## Introduction

### Khroma Coder Executable

Khroma Coder is a program capable of encoding any file into a video and reversing the process afterwards. It is especially useful to be able to send files through platforms that do not trust executables or upload files to platforms that are oriented only to upload videos.

Yet the main rationale for this program is simply experimentation and the visualization and storage of data in alternative ways to the traditional ones.

The program has several modes in which the program can encode the data in the final video file. It must be considered that each modality requires an optimum video quality so that once the file has been decoded it has not been corrupted during the process due to loss of quality.

In version 1.0.17 not all modes are available (more details shortly). All modalities (Referred as “densities” in the executable) to which the program is oriented are the following:

- **Mono (1bppx)**: This mode encodes a single bit per pixel. And should be the most tolerant to any quality-loss in the stored video. This mode is disabled by the reasons stated in the **Considerations** section.
- **Dual (2bppx)**: This mode encodes two bits per pixel. This mode is disabled by the reasons stated in the **Considerations** section.
- **Halfbyte (4bppx)**: This mode effectively encodes half-byte in each pixel. Differing from the most dense methods this one refers to a color code chart explained in its own category on this document.
- **2 Bytes (16bppx)**: This mode encodes two bytes in each pixel. It shares a lot of mechanics in common with the 3 Bytes (24bppx) mode since in some cases the 3 Bytes mode fallback to using 2 Bytes mode for certain pixels. These mechanics are stated in its own category on this document as well.
- **3 Bytes (24bppx)**: This mode encodes three bytes in each pixel. Is the most dense mode supported by this program and requires the highest video quality since even the smallest change can dramatically corrupt the resultant file.



## Khroma Coder User Interface

The Khroma Coder interface is operated using the arrow, enter and escape keys.

The Escape key is used to cancel any action and/or exit the menus but note that in certain cases such as encoding/decoding processes you must keep it pressed for the program to read that it has been pressed.

Note that because it is a console application written in C# the window is always considered active so it actively and constantly registers the keystrokes; that means that if you press the keystrokes, the window is always active. So even if you keep Khroma Coder minimized and you press any of the keys that the executable uses it will be listened and will act accordingly.

There's no easy workaround for this since the way that C# console applications and processes work is very different from a standard Windows application so keep that in mind.

## Khroma Coder Output and Configuration

Khroma Coder **does not save any of the configurations set by the user** so you must change them on the fly in order to enable them. In the same way the output folders are fixed and cannot be changed in any way. The needed folders are; *KCFileOUT*, *KCTemporaryFiles*, *KCVideoOUT*.

**Any encoded file will output its video counterpart into *KCVideoOut* and any decoded video will output its original file counterpart into *KCFileOut*.** *KCTemporaryFiles* folder is where the frames are stored before being converted into a video. Then they're automatically deleted.

## Khroma Coder Data Structure

The data structure used by Khroma Coder is static in all its modalities. Only the encoding in which the data is embedded changes. Thus, the data pattern that Khroma Coder follows is fixed and uses markers or headers of black (**0**) and white (**1**) color combinations.

These colors are reserved only for this purpose and no other encoding uses them, causing relatively complex mechanics in the *16bppx* and *24bppx* encodings in order to avoid the appearance of these colors.

Certain data structures are only present in the first frame; the header that determines what type of density is the video that the program is going to start reading and other spacers such as those that frame the original file name and extension. Other structures are fixed in all frames and serve to facilitate the monitoring of the file decoding process.



## Data Structure Schematics

### HEADER TYPES

- **1010 0001** - MONO (1bppx)
- **1010 0011** - DUAL (2bppx)
- **1010 0101** - HALFBYTE (4bppx)
- **1010 1101** - 2 BYTES (16bppx)
- **1010 1111** - 3 BYTES (24bppx)

### FRAME STRUCTURE

#### FIRST FRAME STRUCTURE

- 1010 1111** - MAIN HEADER
  - { ... } - COMPLETE FILE NAME AND EXTENSION
  - 1001** - SEPARATOR MARKER
  - { ... } - FRAME NUMBER
  - 1010** - SEPARATOR MARKER
  - { ... } - X SIZE
  - 1011** - SEPARATOR MARKER
  - { ... } - Y SIZE
  - 1101** - SEPARATOR MARKER
  - { ... } - FILE DATA
  - 1111** - SEPARATOR MARKER
  - { ... } - FRAME CHECKSUM VALUE

#### IN-BETWEEN FRAME STRUCTURE

- 1101** - SEPARATOR MARKER
- { ... } - FRAME NUMBER
- 1101** - SEPARATOR MARKER
- { ... } - FILE DATA
- 1111** - SEPARATOR MARKER
- { ... } - FRAME CHECKSUM VALUE



## LAST FRAME STRUCTURE

1101 - SEPARATOR MARKER  
{ ... } - FRAME NUMBER  
1101 - SEPARATOR MARKER  
{ ... } - FILE DATA  
1010 - END OF FILE DATA SEPARATOR MARKER  
0000 - ZERO FILL  
1111 - SEPARATOR MARKER  
{ ... } - FRAME CHECKSUM VALUE

Note that it is perfectly possible that if the encoded file is too short for the last frame structure to be present in the first frame structure thus being a single frame video and skipping the in-between frames ordeal.

## Khroma Coder Encoding Modes

As explained above, Khroma Coder can use different modes to encode the files in the video. These modes are referred to as "**densities**" in the program since they determine what data density each pixel will have.

Besides the black and white colors which are reserved for markers the **Magenta** color (**R = 255, G = 0, B = 255**) is restricted since its the value used as null therefore the executable avoids reading any magenta pixel.

▣ - VOID VALUE (RESTRICTED USE)

## Mono (1bppx) Density

Mono density uses the Green and Blue color channels to encode a single bit in each pixel. This makes it resistant to loss of quality in the final video. However, **this mode is disabled internally** due to discrepancies with FFMPEG codecs (more information in the **Considerations** section).





▣ - RESERVED FOR MARKERS  
▣ - RESERVED FOR MARKERS  
▣ - VALUE FOR {1}  
▣ - VALUE FOR {0}

Example: { **R = 0, G = 0, B = 255** } = Value {0}



## Dual (2bppx) Density

















Dual density uses the Green and Blue color channels to encode two bits in each pixel. Although its existence is obvious due to its simplicity it has no practical use in that it is better than simply using the Mono mode or focusing on the higher density modes. Because of this and for similar circumstances to those surrounding Mono mode, **this mode is disabled internally** (more information in the Considerations section).

-  - RESERVED FOR MARKERS
-  - RESERVED FOR MARKERS
-  - VALUE FOR BIT A {1/0}
-  - VALUE FOR BIT B {1/0}

**Example: { R = 0, G = 255, B = 0 } = Value {01}**

## Half-byte (4bppx) Density

Half-Byte density offers an optimal balance between resistance to video quality loss and the density of data encoded in the video. Even so, the required image quality is high enough that it is still advisable to use the higher data density modes. Unlike the higher encoding modes, this one still uses a fixed color table to encode 4 bits in each color.

-  - RESERVED FOR MARKERS
-  - RESERVED FOR MARKERS / DOUBLED AS {0000}
-  - ( 0, 0, 120) VALUE {0001}
-  - ( 0, 0, 255) VALUE {0010}
-  - ( 0, 120, 0) VALUE {0011}
-  - ( 0, 120, 120) VALUE {0100}
-  - ( 0, 120, 255) VALUE {0101}
-  - ( 0, 255, 0) VALUE {0110}
-  - ( 0, 255, 120) VALUE {0111}
-  - ( 0, 255, 255) VALUE {1000}
-  - (120, 0, 0) VALUE {1001}
-  - (120, 0, 120) VALUE {1010}
-  - (120, 0, 255) VALUE {1011}
-  - (120, 120, 0) VALUE {1100}
-  - (120, 120, 120) VALUE {1101}
-  - (120, 120, 255) VALUE {1110}



■ - (120, 255, 0) VALUE {1111}

**Example: { R = 120, G = 0, B = 255 } = Value {1011}**

## 2 Bytes (16bppx) Density

The 2 Bytes per pixel density is the first one that does not depend on a fixed color table but encodes the value of each byte within the byte value of the Green and Blue channels respectively. Thus being very susceptible to any loss of quality caused in the final video as it depends on the accuracy of the colors to be able to reverse the encoding.

■ - RESERVED FOR MARKERS  
■ - RESERVED FOR MARKERS / DOUBLED AS {0x00/0x00}  
■ - BYTE A VALUE {0x00/0xFF}  
■ - BYTE B VALUE {0x00/0xFF}

**Example: { R = 0, G = 171, B = 233 } = Value {0xAB & 0xE9}**

## 3 Bytes (24bppx) Density

The density of 3 bytes per pixel is the densest that Khroma Coder can offer. It requires a resulting video without any loss of quality, so the resulting video usually occupies the same space as the source file.

It is also the most complex since in order to avoid White and Magenta color in certain cases it automatically jumps to 16bppx (and even 8bppx) density for the necessary pixels.

■ - RESERVED FOR MARKERS  
■ - RESERVED FOR MARKERS / DOUBLED AS {0x00/0x00/0x00}  
■ - BYTE A VALUE {0x00/0xFF}  
■ - BYTE B VALUE {0x00/0xFF}  
■ - BYTE C VALUE {0x00/0xFF}

### The main exceptions are:

If Byte A = 0xFF (255)	Byte A value is moved to the Green
If Byte B = 0xFF (255)	channel and Byte B value is moved to
If Byte C = 0xFF (255)	the Blue channel. Red is kept to {0}.

If Byte A = 0xFF (255)	Byte A value is moved to the Green
If Byte B = 0x00 ( 0)	channel and both Red and Blue are kept
If Byte C = 0xFF (255)	to {0}. (Green transfers to Blue)



If Byte A = 0xFF (255)	Byte A value is moved to the Green
If Byte B = 0x?? (???)	channel and Byte B value is moved to
If Byte C = 0x?? (???)	the Blue channel. Red is kept to {0}.

These exceptions make sure that in no case is the color White or Magenta reproduced accidentally. Similarly the last exception is used for the special method of embedding a single Byte in the pixel used for when there is only 1 Byte left to.

**When the Red channel equals 0xFF (255) this means that there's a single Byte stored into that pixel and that Byte is always stored in the Blue Channel.**

**Example: { R = 255, G = 0, B = 140 } = Value {0x8C}**

**Example: { R = 254, G = 132, B = 241 } = Value {0xFE & 0x84 & 0xF1}**

**Example: { R = 0, G = 195, B = 75 } = Value {0xC3 & 0x4B}**

## Considerations

### Mono & Dual Density Modes

The main goal of the lower density modes is to be able to embed the video into any other video since Khroma Coder is able to read only from the area where it finds compatible data. Thing is I haven't found yet a good FFmpeg balance in order to make the Mono and Dual modes fully functional; They work but they need high quality video almost as the 24bppx and 16bppx modes.

So until someone finds a way to make them work in really low quality videos in a reliable way they will be kept disabled. In order to enable them you only need to uncomment a single line stored in the keyboard logic of the options menu interface function.

Also those lower-density modes rely heavily on tolerance indexes and euclidean comparisons in order to output its data and it's something I can't find the time right now to balance properly in order to make it work as intended.

### Encoding Very Big Files

Encoding very big files can be time consuming due that the software stills in need of some level of optimization. This doesn't means that won't work; it'll just take some time to be able to Encode / Decode big files (Such a big .zip)

**Note:** Since version 1.0.18 encoding small files shouldn't create any sort of errors.





***If you need help, want to give your opinion or want to improve this software don't hesitate to contact me either by GitHub, my YouTube channel or my Discord (All stated below). Thanks for using my software ~***

<https://www.youtube.com/channel/UCAPoPnVq3sG-NELTmkYIECw>

<https://github.com/KarstSkarn>

*Discord: owain.doomer*