The WTF-File format

Version 1.1

by **Dobler**, **Erlacher**, **Hörschläger**, **Lazzaretti**, **Sanden** and **Ubel**

Contents

1. Introduction

The WTF-File format is a **byte-aligned**, **binary image file format** designed to represent both **static and animated images** in a compact and extensible way. It defines a strict layout where every component—fixed or variable in size—is stored in complete bytes, simplifying parsing and reducing implementation complexity.

Each WTF file begins with a **1-byte version field** and a **9-byte header**, followed by a **flexible set of sections** including a **Color Lookup Table (CLUT)**, **metadata**, and **image data**. All fields are modular and aligned to byte boundaries, enabling deterministic reading without the need for bit-level operations.

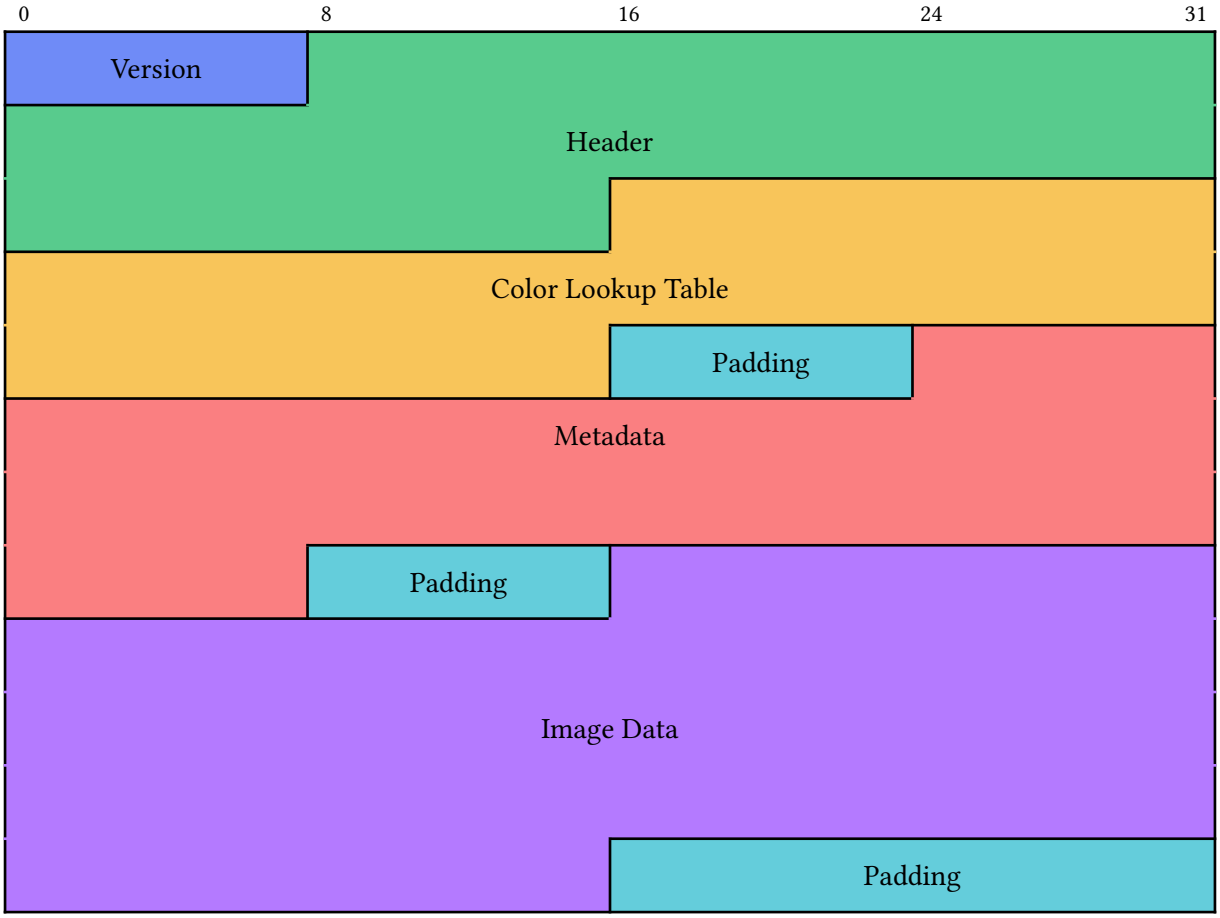
The format supports:

- Up to **65,535 × 65,535 pixels** per image
- **Five color spaces**: Grayscale, RGB, CMY, HSV, and YCbCr
- **Three transparency modes**: None, fixed, and per-pixel (dynamic)
- **Animation** with up to 255 frames and frame timing ranging from 1 frame every 127 seconds to 127 frames per second
- **Compression** via a optional CLUT for image data

Metadata is included using key-value pairs, where keys are restricted to non-control ASCII characters and values may contain arbitrary UTF-8 text. Padding bits are used throughout to maintain byte alignment and structural consistency.

The WTF-File format's strict byte alignment and fixed structure ensure efficient parsing with minimal overhead. Each segment is designed to be processed sequentially or mapped directly into memory, eliminating the need for bit-level operations. This makes the format ideal for low-level systems, embedded environments, custom renderers, or situations where precise control over file structure, memory usage, and data access speed is required.

2. Image Format



Bytefield 1: Image Format

2.1. File Structure

The file consists of multiple segments. Only the Version (1 byte) and Header (9 bytes) have fixed sizes; all other sections are variable in length, depending on the image's configuration (e.g., color mode, metadata, compression).

All fields are byte-aligned and stored as whole bytes. Even for dynamically sized segments, the format guarantees that no bit-level packing occurs. Padding is used as needed to ensure alignment between sections.

2.1.1. General Layout:

1. **Version (1 byte)**: Indicates the format version.
2. **Header (9 bytes)**: Contains configuration data such as color space, transparency mode, and compression type.
3. **Color Lookup Table (CLUT) (variable size)**: A palette for efficient color storage.
4. **Padding (as needed)**: Ensures proper alignment before the next section.
5. **Metadata (variable size)**: Contains optional fields like author, creation date, and version info.
6. **Padding (as needed)**: Alignment padding if required.
7. **Image Data (variable size)**: Raw or compressed pixel data.
8. **Final Padding (as needed)**: Ensures the full structure is byte-aligned.

2.2. Size

The format supports images with up to 65,535 pixels in both width and height.

2.3. Colors

Five color spaces are available, each with three transparency modes.

2.3.1. Supported Color Spaces

- Grayscale
- RGB
- CMY
- HSV
- YCbCr

2.3.2. Variations

- **no transparency**: Standard image without alpha channel
- **fixed transparency**: Single alpha value for the entire image
- **dynamic transparency**: Per-pixel alpha values for full control

2.4. Animation

Animated images can contain up to 255 frames. The frame rate supports a wide range — from 1 frame every 127 seconds up to 127 frames per second — resulting in 254 unique timing options for smooth or slow animations.

2.5. Compression

Colors are compressed using a Color Lookup Table (CLUT). Additional compression may be applied to the image data depending on the settings in the header. The available options are:

- no additional compression
- Run-length encoding (RLE) on image data for further size reduction

2.6. Metadata

Metadata keys (e.g., “author”, “creation date”, “version”) are restricted to non-control ASCII characters. Metadata values, on the other hand, may include any valid UTF-8 characters, allowing for a wide range of localized or detailed information.

3. Version

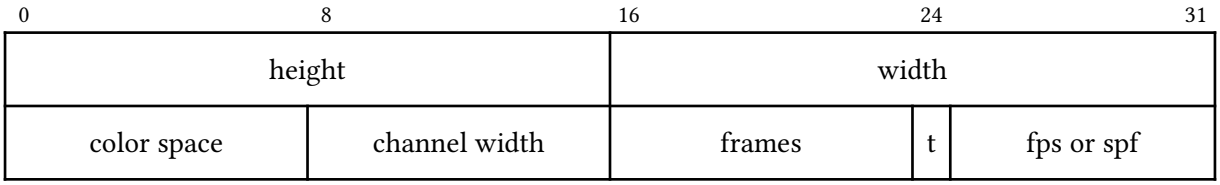
The first field holds a single byte describing the version number. If this field holds a number a viewer or editor can't handle, the image cannot be opened. If the version number is unknown, the image is broken.

Current version: 1

Everything in this specification is only regarding the current version.

4. Header

The header stores some information used to decode the rest of the image. Therefore, the header does not store actual information about the image data, just meta information for the image.



Bytefield 2: Header

4.1. Size (`height` and `width`)

The image size is defined using two fields: height and width. These fields specify the dimensions of the image in pixels.

4.1.1. Field Names

- `height`: number of pixels vertically
- `width`: number of pixels horizontally

4.1.2. Data Format

Both fields are stored as unsigned 16-bit integers.

4.1.3. Constraints

- minimum value: 1
- maximum value: 65 535 ($2^{16} - 1$)

4.2. Color (`color space` and `channel width`)

May define channels as

- `FIXED` have always the same length or
- `DYNAMIC` have length of channel width.

The first digit defines the color space and the second one a variation.

The format supports 5 distinct color spaces with 3 variations each.

4.2.1. Color spaces

- $0x_{16}$ Gray scale
- $1x_{16}$ RGB
- $2x_{16}$ CMY
- $3x_{16}$ HSV
- $4x_{16}$ YCbCr

4.2.2. Variations

- $x0_{16}$ no transparency
- $x1_{16}$ fixed transparency
- $x2_{16}$ dynamic transparency

4.2.3. Constraints

minimum: 1 Bit per channel

maximum: 16 Bit per channel

Code	Color space	Channels
00_{16}	Gray Scale	• `G`, `DYNAMIC`, gray
01_{16}	Transparenten Gray Scale	• `G`, `DYNAMIC`, gray • `a`, `FIXED(1)`, transparent
02_{16}	Dynamic Transparent Gray Scale	• `G`, `DYNAMIC`, gray • `a`, `DYNAMIC`, transparent
10_{16}	RGB	• `R`, `DYNAMIC`, red • `G`, `DYNAMIC`, green • `B`, `DYNAMIC`, blue
11_{16}	RGBa	• `R`, `DYNAMIC`, red • `G`, `DYNAMIC`, green • `B`, `DYNAMIC`, blue • `a`, `FIXED` (1), transparent
12_{16}	Dynamic RGBa	• `R`, `DYNAMIC`, red • `G`, `DYNAMIC`, green • `B`, `DYNAMIC`, blue • `a`, `DYNAMIC`, transparent
20_{16}	CMY	• `C`, `DYNAMIC`, cyan • `M`, `DYNAMIC`, magenta • `Y`, `DYNAMIC`, yellow
21_{16}	CMYa	• `C`, `DYNAMIC`, cyan • `M`, `DYNAMIC`, magenta • `Y`, `DYNAMIC`, yellow • `a`, `FIXED` (1), transparent
22_{16}	Dynamic CMYa	• `C`, `DYNAMIC`, cyan • `M`, `DYNAMIC`, magenta • `Y`, `DYNAMIC`, yellow • `a`, `DYNAMIC`, transparent
30_{16}	HSV	• `H`, `DYNAMIC`, hue • `S`, `DYNAMIC`, saturation • `V`, `DYNAMIC`, value
31_{16}	HSVa	• `H`, `DYNAMIC`, hue • `S`, `DYNAMIC`, saturation

Code	Color space	Channels
		• `V`, `DYNAMIC`, value • `a`, `FIXED` (1), transparent
32_{16}	Dynamic HSVa	• `H`, `DYNAMIC`, hue • `S`, `DYNAMIC`, saturation • `V`, `DYNAMIC`, value • `a`, `DYNAMIC`, transparent
40_{16}	YCbCr	• `Y`, `DYNAMIC`, lumina • `Cb`, `DYNAMIC`, Chroma blue • `Cr`, `DYNAMIC`, Chroma red
41_{16}	YCbCra	• `Y`, `DYNAMIC`, lumina • `Cb`, `DYNAMIC`, Chroma blue • `Cr`, `DYNAMIC`, Chroma red • `a`, `FIXED` (1), transparent
42_{16}	Dynamic YCbCra	• `Y`, `DYNAMIC`, lumina • `Cb`, `DYNAMIC`, Chroma blue • `Cr`, `DYNAMIC`, Chroma red • `a`, `DYNAMIC`, transparent

Table 1: Color spaces

4.3. Animation (`frames`, `t`, and `fps or spf`)

Images may be animated. The animation is defined by an amount of frames, and a timing for the frames.

4.3.1. Field Names

- `frames`: the amount of frames the image has
- `t`: the unit of the following timing
- `fps or spf`: the timing of the frames

4.3.2. Data Format

`frames` is stored as an unsigned 8-bit integer.

`t` is stored as a 1-bit flag. 0 indicates, that the following value is the amount of frames that should be displayed per second (fps). Therefore, a frame must stay $\frac{1}{\text{fps}}$ seconds. 1 indicates, that the given value is the amount of seconds a frame should be displayed (spf).

`fps or spf` is stored as a unsigned 7-bit integer.

4.3.3. Constraints

4.3.3.1. `frames`

- minimum value: 1
- maximum value: 255 ($2^8 - 1$)

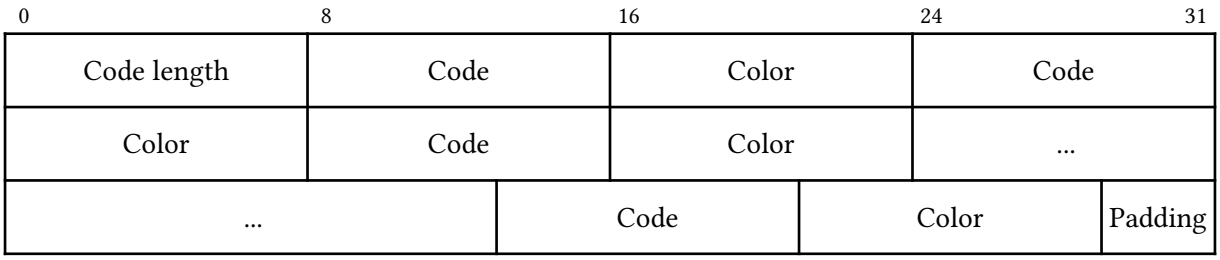
4.3.3.2. `t`

Must be 0, if `frames` is 1.

4.3.3.3. `fps or spf`

- exact value: 0, if `frames` is 0.
- minimum value: 1, if `frames` ≥ 1
- maximum value: 127 ($2^7 - 1$), if `frames` ≥ 1

5. Color Lookup Table



Bytefield 3: Color Lookup Table

Map of Code to Color (Channels * Channel width), ended by code only containing 0.

5.1. Structure

- `Code length`: Amount of bits used for the following code. Code length depends on the amount of colors in the picture. It's mentioned in the first byte.
- `Code`: Sequence of bits representing the subsequent color.
- `Color`: Color in set color space with set color width.
- `Padding`: Padded with bits with value 0 to full byte.

5.1.1. Example Color Look-Up Table

- Code length = 00000011_2
- Colors in set color space
- 1 byte colors in hex in the example

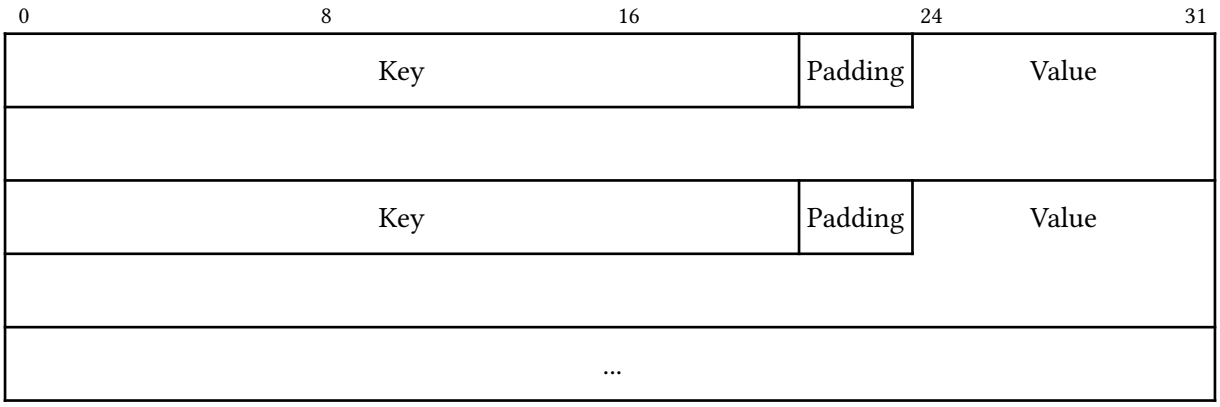
Code	Color
001_2	004078
010_2	0080FF
011_2	A0B312
100_2	F31818
101_2	929292
110_2	C4C4C4
111_2	FFFFFF
000_2	*Termination*

Table 2: Example Color Lookup Table

5.2. Constraints

min. entries: 1 max. entries: 2^{64}

6. Metadata



Bytefield 4: Metadata

The metadata section stores descriptive information as a sequence of key-value pairs. This allows embedding of human-readable or machine-readable information, such as author name, creation date, software version, etc.

6.1. Structure

- Metadata consists of alternating `key` and `value` entries.
- Each `key` is:
  - A string of 7-bit ASCII characters: a–z, A–Z, 0–9, -
  - Null-terminated (0_{16})
  - Padded with bit with value 0 to a full byte
- Each `value` is:
  - A UTF-8 encoded string
  - Null-terminated (0_{16})
- The sequence may be repeated for multiple key-value pairs.

6.2. Termination

The metadata section is terminated by a single null byte 0_{16} representing an empty key, followed by a single byte of padding, with no corresponding value.

6.3. Constraints

- `Keys` must not contain the NUL-character.
- `Values` may include any valid UTF-8 characters except the NUL-character.

7. Image Data

An image consists of $f \cdot w \cdot h$ pixels, where

- f is the number of frames (see Section 4.3) and
- w and h are the width and the height of the image (see Section 4.1).

The pixels are stored in a chain one after another, starting with the top left pixel of the first frame, going from left to right and top to bottom.

7.1. Example pixel positions

Let $f = 3, w = 5$ and $h = 3$, then the pixels would be order as follows:

1	**2**	**3**	**4**	**5**
6	7	8	9	10
11	12	13	14	15

Table 3: Example Frame 1

16	**17**	**18**	**19**	**20**
21	22	23	24	25
26	27	28	29	30

Table 4: Example Frame 2

31	**32**	**33**	**34**	**35**
36	37	38	39	40
41	42	43	44	45

Table 5: Example Frame 3

7.2. Pixel data

Each pixel is a combination of a *3 bit* pixel type and the actual pixel data. There are the following pixel types:

Code	Type
000_2	Color entry
001_2	Copy by location
010_2	Copy by frame
011_2	Copy by frame and location
100_2	Copy previous location
101_2	Copy previous frame
110_2	Color lookup table Entry
111_2	*Currently not used*

Table 6: Pixel types

7.2.1. Color entry

A color entry consists of an actual color entry.

The length l of the entry (in bits) is $l = w \cdot d + s$ where w is the channel width, d is the number of dynamic channels defined by the color space and s is the number of fixed channels defined by the color space (see Section 4.2).

The single channel entries are given in the same order as specified in the specification.

7.2.2. Copy entry

A copy entry (001_2 - 101_2) copies the color entry of another pixel. That pixel may be of any type. However, the pixel entry must at some point be a “Color entry” (000_2).

A copy by frame (and location) is followed by the frame. A copy by frame and location is followed by the x and y coordinates afterward. A copy by location is directly followed by the x and y coordinates.

The number of bits used to store the frame, x or y data are calculated using $\lceil \log_2(a) \rceil$ where a is the maximum value.

Maximum value	Bits
1	0
2	1
3,4	2
5-8	3
9-16	4
⋮	⋮

Table 7: Image size to bits

The copy previous location copies the pixel directly stored before this pixel in binary. If this pixel is the first pixel of a frame, the image is considered broken.

The copy previous frame copies the pixel at the same position as the current pixel from the previous frame. If this pixel is in the first frame, the image is considered broken.

7.2.3. Color lookup table entry

A color lookup table entry type is followed by the exact number of bits for a clut entry. This indicates the clut entry that should be used. See Section 5.

List of tables

List of bytefields