

# WebP Container Specification

---

 [developers.google.com/speed/webp/docs/riff\\_container](https://developers.google.com/speed/webp/docs/riff_container)

## Introduction


---

WebP is an image format that uses either (i) the VP8 key frame encoding to compress image data in a lossy way, or (ii) the WebP lossless encoding (and possibly other encodings in the future). These encoding schemes should make it more efficient than currently used formats. It is optimized for fast image transfer over the network (e.g., for websites). The WebP format has feature parity (color profile, metadata, animation etc) with other formats as well. This document describes the structure of a WebP file.

The WebP container (i.e., RIFF container for WebP) allows feature support over and above the basic use case of WebP (i.e., a file containing a single image encoded as a VP8 key frame). The WebP container provides additional support for:

- **Lossless compression.** An image can be losslessly compressed, using the WebP Lossless Format.
- **Metadata.** An image may have metadata stored in EXIF or XMP formats.
- **Transparency.** An image may have transparency, i.e., an alpha channel.
- **Color Profile.** An image may have an embedded ICC profile as described by the [International Color Consortium](#).
- **Animation.** An image may have multiple frames with pauses between them, making it an animation.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#).

Bit numbering in chunk diagrams starts at  for the most significant bit ('MSB 0') as described in [RFC 1166](#).

## Naming

---

It is RECOMMENDED to use the following types when referring to the WebP container:

Container Format Name	WebP
Filename Extension	.webp
MIME-type	image/webp

# Terminology & Basics

A WebP file contains either a still image (i.e., an encoded matrix of pixels) or an animation. Optionally, it can also contain transparency information, color profile and metadata. In case we need to refer only to the matrix of pixels, we will call it the *canvas* of the image.

Below are additional terms used throughout this document:

### *Reader/Writer*

Code that reads WebP files is referred to as a *reader*, while code that writes them is referred to as a *writer*.

***uint16***

A 16-bit, little-endian, unsigned integer.

***uint24***

A 24-bit, little-endian, unsigned integer.

***uint32***

A 32-bit, little-endian, unsigned integer.

**FourCC**

A *FourCC* (four-character code) is a *uint32* created by concatenating four ASCII characters in little-endian order.

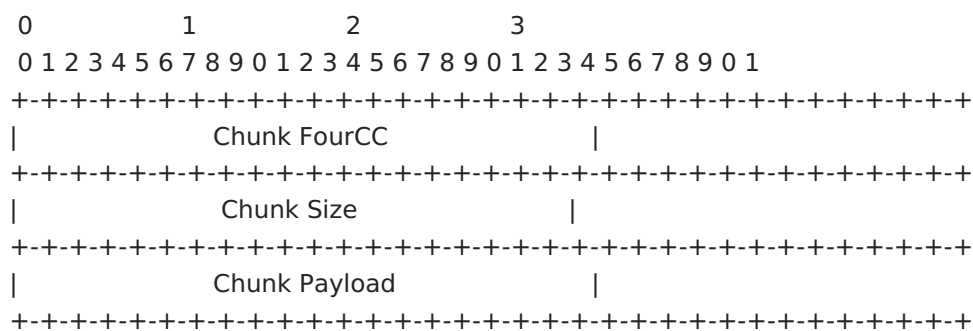
**1-based**

An unsigned integer field storing values offset by **-1** . e.g., Such a field would store value 25 as 24.

# RIFF File Format

The WebP file format is based on the RIFF (resource interchange file format) document format.

The basic element of a RIFF file is a *chunk*. It consists of:



**Chunk FourCC: 32 bits**

ASCII four-character code used for chunk identification.

**Chunk Size: 32 bits (*uint32*)**

The size of the chunk not including this field, the chunk identifier or padding.

**Chunk Payload:** *Chunk Size* bytes

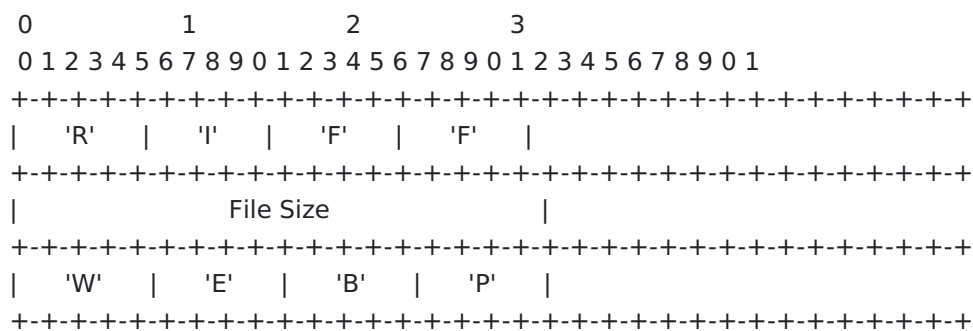
The data payload. If *Chunk Size* is odd, a single padding byte -- that SHOULD be 0 -- is added.

***ChunkHeader('ABCD')***

This is used to describe the *FourCC* and *Chunk Size* header of individual chunks, where 'ABCD' is the FourCC for the chunk. This element's size is 8 bytes.

**Note:** RIFF has a convention that all-uppercase chunk FourCCs are standard chunks that apply to any RIFF file format, while FourCCs specific to a file format are all lowercase. WebP does not follow this convention.

## WebP File Header



**'RIFF': 32 bits**

The ASCII characters 'R' 'l' 'F' 'F'.

**File Size: 32 bits (*uint32*)**

The size of the file in bytes starting at offset 8. The maximum value of this field is  $2^{32}$  minus 10 bytes and thus the size of the whole file is at most 4GiB minus 2 bytes.

**'WEBP': 32 bits**

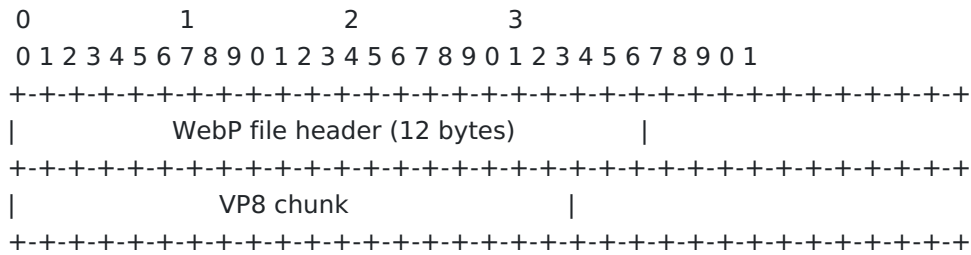
The ASCII characters 'W' 'E' 'B' 'P'.

A WebP file MUST begin with a RIFF header with the FourCC 'WEBP'. The file size in the header is the total size of the chunks that follow plus 4 bytes for the 'WEBP' FourCC. The file SHOULD NOT contain anything after it. As the size of any chunk is even, the size given by the RIFF header is also even. The contents of individual chunks will be described in the following sections.

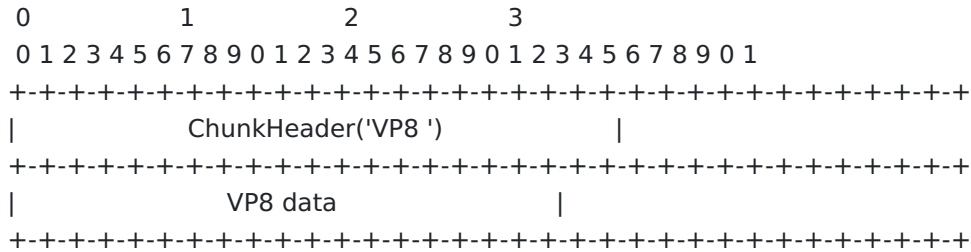
## Simple File Format (Lossy)

This layout SHOULD be used if the image requires *lossy* encoding and does not require transparency or other advanced features provided by the extended format. Files with this layout are smaller and supported by older software.

Simple WebP (lossy) file format:



VP8 chunk:



**VP8 data: *Chunk Size* bytes**

VP8 bitstream data.

The VP8 bitstream format specification can be found at [VP8 Data Format and Decoding Guide](#). Note that the VP8 frame header contains the VP8 frame width and height. That is assumed to be the width and height of the canvas.

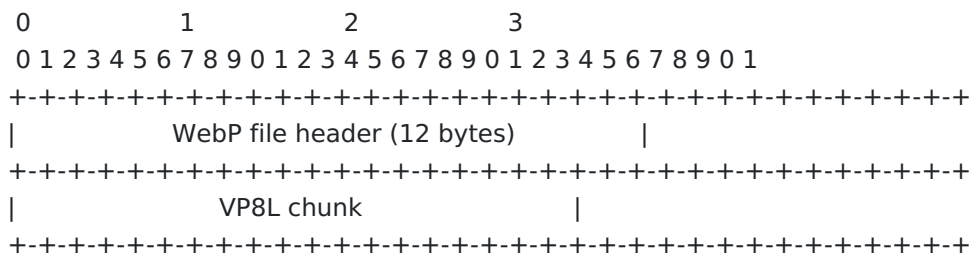
The VP8 specification describes how to decode the image into Y'CbCr format. To convert to RGB, Rec. 601 SHOULD be used.

## Simple File Format (Lossless)

**Note:** Older readers may not support files using the lossless format.

This layout SHOULD be used if the image requires *lossless* encoding (with an optional transparency channel) and does not require advanced features provided by the extended format.

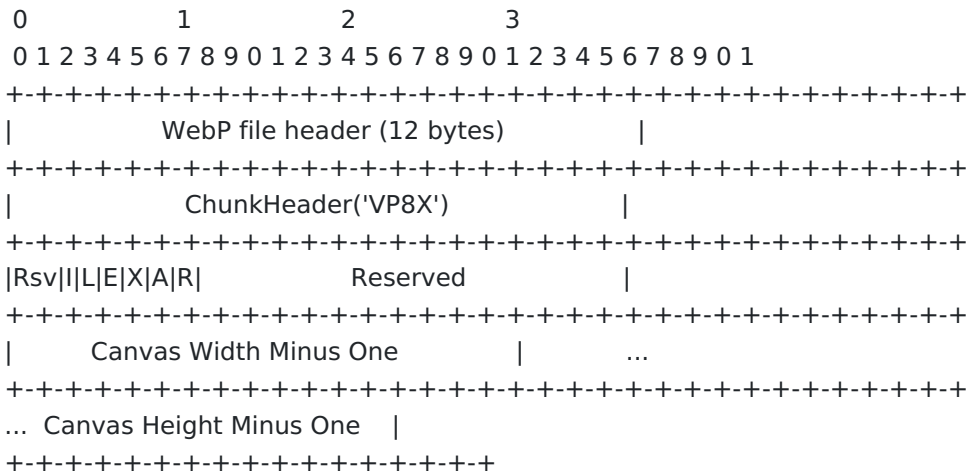
Simple WebP (lossless) file format:



VP8L chunk:



## Extended WebP file header:



**Reserved (Rsv): 2 bits**

SHOULD be 0.

### ICC profile (I): 1 bit

Set if the file contains an ICC profile.

**Alpha (L): 1 bit**

Set if any of the frames of the image contain transparency information ("alpha").

**EXIF metadata (E): 1 bit**

Set if the file contains EXIF metadata.

**XMP metadata (X): 1 bit**

Set if the file contains XMP metadata.

**Animation (A): 1 bit**

Set if this is an animated image. Data in 'ANIM' and 'ANMF' chunks should be used to control the animation.

**Reserved (R): 1 bit**

SHOULD be 0.

**Reserved: 24 bits**

SHOULD be 0.

### Canvas Width Minus One: 24 bits

*1-based* width of the canvas in pixels. The actual canvas width is '1 + Canvas Width Minus One'

### Canvas Height Minus One: 24 bits

1-based height of the canvas in pixels. The actual canvas height is '1 + Canvas Height Minus One'

The product of *Canvas Width* and *Canvas Height* MUST be at most  $2^{32} - 1$ .

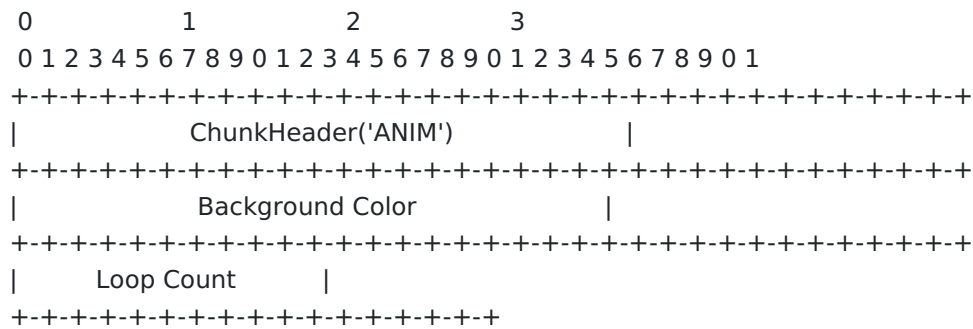
Future specifications MAY add more fields.

# Animation

An animation is controlled by ANIM and ANMF chunks.

ANIM Chunk:

For an animated image, this chunk contains the *global parameters* of the animation.



## Background Color: 32 bits (*uint32*)

The default background color of the canvas in [Blue, Green, Red, Alpha] byte order. This color MAY be used to fill the unused space on the canvas around the frames, as well as the transparent pixels of the first frame. Background color is also used when disposal method is **1** .

### Note:

- Background color MAY contain a transparency value (alpha), even if the *Alpha* flag in VP8X chunk is unset.
- Viewer applications SHOULD treat the background color value as a hint, and are not required to use it.
- The canvas is cleared at the start of each loop. The background color MAY be used to achieve this.

## Loop Count: 16 bits (*uint16*)

The number of times to loop the animation. **0** means infinitely.

This chunk MUST appear if the *Animation* flag in the VP8X chunk is set. If the *Animation* flag is not set and this chunk is present, it SHOULD be ignored.

ANMF chunk:

For animated images, this chunk contains information about a *single* frame. If the *Animation flag* is not set, then this chunk SHOULD NOT be present.





- **0** : Do not dispose. Leave the canvas as is.
- **1** : Dispose to background color. Fill the *rectangle* on the canvas covered by the *current frame* with background color specified in the ANIM chunk.

### Notes:

- The frame disposal only applies to the *frame rectangle*, that is, the rectangle defined by *Frame X*, *Frame Y*, *frame width* and *frame height*. It may or may not cover the whole canvas.
- **Alpha-blending**: Given that each of the R, G, B and A channels is 8-bit, and the RGB channels are *not premultiplied* by alpha, the formula for blending 'dst' onto 'src' is:

```
blend.A = src.A + dst.A * (1 - src.A / 255)
if blend.A = 0 then
    blend.RGB = 0
else
    blend.RGB = (src.RGB * src.A +
                 dst.RGB * dst.A * (1 - src.A / 255)) / blend.A
```

- Alpha-blending SHOULD be done in linear color space, by taking into account the color profile of the image. If the color profile is not present, sRGB is to be assumed. (Note that sRGB also needs to be linearized due to a gamma of ~2.2).

### Frame Data: **Chunk Size** - **16** bytes

\* An optional alpha subchunk for the frame.

A bitstream subchunk for the frame.

**Note:** The 'ANMF' payload, *Frame Data* above, consists of individual *padded* chunks as described by the RIFF file format.

## Alpha

```

0           1           2           3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           ChunkHeader('ALPH')           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Rsv| P | F | C |   Alpha Bitstream...   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

### Reserved (Rsv): 2 bits

SHOULD be **0** .

### Pre-processing (P): 2 bits

These INFORMATIVE bits are used to signal the pre-processing that has been performed during compression. The decoder can use this information to e.g. dither the values or smooth the gradients prior to display.

- 0 : no pre-processing
- 1 : level reduction

### Filtering method (F): 2 bits

The filtering method used:

- 0 : None.
- 1 : Horizontal filter.
- 2 : Vertical filter.
- 3 : Gradient filter.

For each pixel, filtering is performed using the following calculations. Assume the alpha values surrounding the current X position are labeled as:

```
C | B |
---+---+
A | X |
```

We seek to compute the alpha value at position X. First, a prediction is made depending on the filtering method:

- Method 0 : predictor = 0
- Method 1 : predictor = A
- Method 2 : predictor = B
- Method 3 : predictor = clip(A + B - C)

where clip(v) is equal to:

- 0 if  $v < 0$
- 255 if  $v > 255$
- v otherwise

The final value is derived by adding the decompressed value X to the predictor and using modulo-256 arithmetic to wrap the [256-511] range into the [0-255] one:

$$\text{alpha} = (\text{predictor} + X) \% 256$$

There are special cases for left-most and top-most pixel positions:

- Top-left value at location (0,0) uses 0 as predictor value. Otherwise,
- For horizontal or gradient filtering methods, the left-most pixels at location (0, y) are predicted using the location (0, y-1) just above.
- For vertical or gradient filtering methods, the top-most pixels at location (x, 0) are predicted using the location (x-1, 0) on the left.

Decoders are not required to use this information in any specified way.

### Compression method (C): 2 bits

The compression method used:

- **0** : No compression.
- **1** : Compressed using the WebP lossless format.

### Alpha bitstream: *Chunk Size* - **1** bytes

Encoded alpha bitstream.

This optional chunk contains encoded alpha data for this frame. A frame containing a 'VP8L' chunk SHOULD NOT contain this chunk.

**Rationale:** The transparency information is already part of the 'VP8L' chunk.

The alpha channel data is stored as uncompressed raw data (when compression method is '0') or compressed using the lossless format (when the compression method is '1').

- Raw data: consists of a byte sequence of length width \* height, containing all the 8-bit transparency values in scan order.
- Lossless format compression: the byte sequence is a compressed image-stream (as described in the [WebP Lossless Bitstream Format](#)) of implicit dimension width x height. That is, this image-stream does NOT contain any headers describing the image dimension.

**Rationale:** the dimension is already known from other sources, so storing it again would be redundant and error-prone.

Once the image-stream is decoded into ARGB color values, following the process described in the lossless format specification, the transparency information must be extracted from the *green* channel of the ARGB quadruplet.

**Rationale:** the green channel is allowed extra transformation steps in the specification -- unlike the other channels -- that can improve compression.

## Bitstream (VP8/VP8L)

---

This chunk contains compressed bitstream data for a single frame.

A bitstream chunk may be either (i) a VP8 chunk, using "VP8 " (note the significant fourth-character space) as its tag *or* (ii) a VP8L chunk, using "VP8L" as its tag.

The formats of VP8 and VP8L chunks are as described in sections [Simple File Format \(Lossy\)](#) and [Simple File Format \(Lossless\)](#) respectively.

## Color profile

---



## XMP Metadata: *Chunk Size* bytes

image metadata in XMP format.

Additional guidance about handling metadata can be found in the Metadata Working Group's [Guidelines for Handling Metadata](#).

Displaying an *animated image* canvas MUST be equivalent to the following pseudocode:

```
assert VP8X.flags.hasAnimation
canvas ← new image of size VP8X.canvasWidth x VP8X.canvasHeight with
    background color ANIM.background_color.
loop_count ← ANIM.loopCount
dispose_method ← ANIM.disposeMethod
if loop_count == 0:
    loop_count = ∞
frame_params ← nil
assert next chunk in image_data is ANMF
for loop = 0..loop_count - 1
    clear canvas to ANIM.background_color or application defined color
    until eof or non-ANMF chunk
        frame_params.frameX = Frame X
        frame_params.frameY = Frame Y
        frame_params.frameWidth = Frame Width Minus One + 1
        frame_params.frameHeight = Frame Height Minus One + 1
        frame_params.frameDuration = Frame Duration
        frame_right = frame_params.frameX + frame_params.frameWidth
        frame_bottom = frame_params.frameY + frame_params.frameHeight
        assert VP8X.canvasWidth >= frame_right
        assert VP8X.canvasHeight >= frame_bottom
        for subchunk in 'Frame Data':
            if subchunk.tag == "ALPH":
                assert alpha subchunks not found in 'Frame Data' earlier
                frame_params.alpha = alpha_data
            else if subchunk.tag == "VP8 " OR subchunk.tag == "VP8L":
                assert bitstream subchunks not found in 'Frame Data' earlier
                frame_params.bitstream = bitstream_data
    render frame with frame_params.alpha and frame_params.bitstream on
    canvas with top-left corner at (frame_params.frameX,
    frame_params.frameY), using dispose method dispose_method.
canvas contains the decoded image.
Show the contents of the canvas for
    frame_params.frameDuration * 1ms.
```

## Example File Layouts

---

A lossy encoded image with alpha may look as follows:

```
RIFF/WEBP
+- VP8X (descriptions of features used)
+- ALPH (alpha bitstream)
+- VP8 (bitstream)
```

A losslessly encoded image may look as follows:

RIFF/WEBP

- + - VP8X (descriptions of features used)
- + - VP8L (lossless bitstream)

A lossless image with ICC profile and XMP metadata may look as follows:

RIFF/WEBP

- + - VP8X (descriptions of features used)
- + - ICCP (color profile)
- + - VP8L (lossless bitstream)
- + - XMP (metadata)

An animated image with EXIF metadata may look as follows:

RIFF/WEBP

- + - VP8X (descriptions of features used)
- + - ANIM (global animation parameters)
- + - ANMF (frame1 parameters + data)
- + - ANMF (frame2 parameters + data)
- + - ANMF (frame3 parameters + data)
- + - ANMF (frame4 parameters + data)
- + - EXIF (metadata)