International Standard

ISO/IEC 23001-17

First edition
2024-02

Information technology — MPEG systems technologies —

Part 17: Carriage of uncompressed video and images in ISO base media file format

Technologies de l'information — Technologies des systèmes MPEG —

Partie 17: Transport de vidéos et images non compressées dans le format ISO de base pour les fichiers médias

Reference number
ISO/IEC 23001-17:2024(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia, and hypermedia*.

A list of all parts in the ISO/IEC 23001 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Information technology — MPEG systems technologies —

Part 17: Carriage of uncompressed video and images in ISO base media file format

1 Scope

This document specifies how uncompressed 2D image and video data is carried in files in the family of standards based on the ISO base media file format (ISO/IEC 14496-12). This includes but is not limited to monochromatic data, colour data, transparency (alpha) information and depth information.

The primary goal of this document is to allow exchange of uncompressed video and image data while relying on the information set provided by the ISO base media file format, such as timing, colour space and sample aspect ratio to specify the interpretation and/or display of video and image data.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 14496-12, *Information technology — Coding of audio-visual objects — Part 12: ISO base media file format*

ISO/IEC 23008-12, *Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 12: Image File Format*

IEEE 754-2008, *IEEE Standard for Floating-Point Arithmetic*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 14496-12 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
block
consecutive bytes within the sample data containing one or more component values for one or more pixels and possible padding

3.2
component
part of the image data representing a single channel (or dimension) of the image

Note 1 to entry: In this document, a component may describe visual information such as luminance or chroma, or other information usually not intended for direct display such as depth or transparency.

3.3
frame
two-dimensional rectangular array of pixels contained in the sample data

3.4
interleaving
ordering of pixel or component within the sample data

3.5
pixel
smallest element of an image, comprised of one or more components

3.6
row
horizonal line of pixels within a frame or a tile

3.7
sample data
payload of the media sample when the uncompressed frame is described by a media track, or payload of the item when the uncompressed frame is described by an image item

Note 1 to entry: Media sample as defined in ISO/IEC 14496-12.

Note 2 to entry: Image item as defined in ISO/IEC 23008-12.

3.8
tile
two-dimensional rectangular array of pixels within a frame

3.9
uncompressed frame
frame for which each value of each component is coded independently from any other component value in the same frame or any other frame

Note 1 to entry: In this document, the uncompressed term is used with some video formats applying sub-sampling of some components for the purpose of data reduction; however, data access to each individual component for such formats is still independent from other components or frames.

3.10
uncompressed image
single uncompressed frame stored as an image item

3.11
uncompressed video
sequence of one or more uncompressed frames

4 Uncompressed video and image formats

4.1 Overview

Uncompressed frames may be stored in ISO base media files as media samples of media tracks or as image items using a generic uncompressed video description defined in this document.

Media tracks, media samples and image items may be further described using the various tools defined in ISO/IEC 14496-12, such as sample group descriptions, `MetaBox`, metadata tracks and sample auxiliary information. User-defined components may be used to carry per-pixel specific information, either in the same sample or item as the described pixels or in a separate track or item.

The tools defined in ISO/IEC 14496-12 and ISO/IEC 23008-12 should be used whenever applicable, namely to specify pixel aspect ratio, colour information, clean aperture, content light level, mirror and rotate properties or track header matrix, etc.

An uncompressed video media sample or item consists of one uncompressed frame. Each uncompressed frame is organized as a set of one or more rectangular, non-overlapping and contiguous (without holes) areas called tiles.

ISOBMF allows constructing files referring to external data. This enables building ISOBMF files describing existing uncompressed image or video files without having to copy the media data, simplifying integration into existing workflows.

4.2 Storage in media tracks

Uncompressed video tracks compliant to this document are tracks compliant to ISO/IEC 14496-12 that use a `VisualSampleEntry` with `codingname` equal to `'uncv'`, hereafter called uncompressed video sample entry.

The uncompressed video sample entry shall contain

— one `UncompressedFrameConfigBox`;

— one `ComponentDefinitionBox` if the `UncompressedFrameConfigBox` version is not 1.

When both `UncompressedFrameConfigBox` and `ComponentDefinitionBox` are present in the sample entry, the `ComponentDefinitionBox` shall precede the `UncompressedFrameConfigBox`.

The `compressorname` field of an uncompressed video sample entry should be set to all 0 (empty string). The `depth` field of an uncompressed video sample entry shall be ignored and should be set to 0, the bit depth per component being indicated by the `UncompressedFrameConfigBox`.

The handler type associated with the track is usually `'vide'`, `'auxv'` or `'pict'` but derived specifications may introduce new handler types. The `width` and `height` fields of the sample entry documents the exact frame dimension, in pixels of any non-subsampled component, of any sample of the video stream that is described by this sample entry. Consequently, if the frame dimension changes within a video track, multiple sample entries shall be used.

The payload of an uncompressed video media sample consists of one uncompressed frame.

The size in bytes of an uncompressed video media sample shall be at least the size in bytes required to store all the components values documented by the uncompressed video configuration as defined in subclause 5.2.

NOTE The sample data can be larger than the size in bytes required to store all the components values, typically to store information in the trailing data. How such additional bytes are handled by a file reader is out of scope of this document.

Each uncompressed video media sample shall be marked as a sync sample. As a consequence, the `SyncSampleBox`, `ShadowSyncSampleBox`, `CompositionOffsetBox` and `CompositionToDecodeBox` shall not be present in the track.

Media tracks containing only non-visual components should be marked as not present in the presentation, i.e. `track_in_movie` flag should not be set.

Media tracks containing user-defined components providing per-pixel information for pixels in another track should use a track reference of type `'cdsc'` to the track they describe.

4.3 Storage in image items

An uncompressed image compliant to this document is an image item compliant to ISO/IEC 23008-12 with the `item_type 'unci'`.

An uncompressed image shall be associated with:

— an `UncompressedFrameConfigBox` essential item property, i.e., `essential` shall be equal to 1 for an `UncompressedFrameConfigBox` item property associated with an image item of type `'unci'`;

— a `ComponentDefinitionBox` essential item property if the `UncompressedFrameConfigBox` version is not 1;

— an `ImageSpatialExtentsProperty` whose `image_width` and `image_height` fields shall document the exact frame dimension, in pixels, of the reconstructed image, i.e. the size of the image before applying any associated transformative properties.

The payload of an uncompressed image consists of one uncompressed frame.

The size in bytes of an uncompressed image item shall be at least the size in bytes required to store all the components values documented by the uncompressed video configuration as defined in subclause 5.2.

NOTE The image item data can be larger than the size in bytes required to store all the components values, typically to store information in the trailing data. How such additional bytes are handled by a file reader is out of scope of this document.

An uncompressed image can be further documented using the various tools defined in ISO/IEC 14496-12 or ISO/IEC 23008-12, such as descriptive item properties.

Uncompressed images containing only non-visual components should be marked as hidden items, i.e. have (`flags` & 1) equal to 1 in their `ItemInfoEntry`.

Uncompressed images containing user-defined components providing per-pixel information for pixels in another item should use an item reference of type `'cdsc'` to the item they describe.

If an uncompressed item is associated with an `AuxiliaryTypeProperty` indicating alpha (resp. depth), the uncompressed item shall have one and only one component of type alpha (resp. depth).

5 Uncompressed frame description

5.1 Component Definition

5.1.1 Definition

Box Type: `'cmpd'`
Container: Video sample entry, `ItemPropertyContainerBox`
Mandatory: see below
Quantity: At most one per uncompressed video sample entry or at most one associated per uncompressed image item

The `ComponentDefinitionBox` is used to document the types of components present in samples or items associated with this box through the sample entry or through item property association.

Components defined in the `ComponentDefinitionBox` are referenced by indexes in various boxes in this document. Care has to be taken while removing components from an uncompressed video or image to also remove in other boxes any reference to the removed components. There is no requirement that all components defined in the `ComponentDefinitionBox` are referenced by other boxes.

For all boxes referring to components defined in the `ComponentDefinitionBox`, the associated `ComponentDefinitionBox` (resp. the associated `UncompressedFrameConfigBox`) is defined as:

— for an uncompressed video, the `ComponentDefinitionBox` (resp. `UncompressedFrameConfigBox`) present in the same sample entry as the referring box;

— for an uncompressed image, the `ComponentDefinitionBox` (resp. `UncompressedFrameConfigBox`) associated, through `ItemPropertyAssociationBox`, to the same image item as the referring box.

This box shall be present if the version of the associated `UncompressedFrameConfigBox` is 0.

This box may be present if the version of the associated `UncompressedFrameConfigBox` is 1.

When the `ComponentDefinitionBox` is not present, the associated `ComponentDefinitionBox` is implicitly defined as indicated by the profile of the associated `UncompressedFrameConfigBox`, see subclause 5.3.

The component_index field value defined in boxes using an associated ComponentDefinitionBox indicates the index in the list of components, with value 0 indicating the first component listed in the associated ComponentDefinitionBox. The component_index field value shall be strictly less than the component_count field value of the associated ComponentDefinitionBox.

A ComponentDefinitionBox may describe two or more components with the same type (e.g. two monochrome components representing different portions of the electromagnetic spectrum), and file readers may require additional information to process the pixel data. How such information is provided to the file reader is out of scope of this document.

Table 1 — Component types

Value	Description
0	Monochrome component
1	Luma component (Y)
2	Chroma component (Cb / U)
3	Chroma component (Cr / V)
4	Red component (R)
5	Green component (G)
6	Blue component (B)
7	Alpha/transparency component (A)
8	Depth component (D)
9	Disparity component (Disp)
10	Palette component (P) The component_format value for this component shall be 0.
11	Filter Array (FA) component such as Bayer, RGBW, etc.
12	Padded component (unused bits/bytes)
13	Cyan component (C)
14	Magenta component (M)
15	Yellow component (Y)
16	Key (black) component (K)
17 to 0x7FFF	ISO/IEC reserved
0x8000 to 0xFFFF	User-defined component type(s)

Component types reflect only a nominal characterization of the pixel data and how that pixel data should be displayed; precise bandpass limits, for example, are not implied and may differ from image to image. For some component types, some other boxes can provide additional information, such as ColourInformationBox or MasteringDisplayColourVolumeBox. For other component types, the signalling of such information is out of scope of this document.

if a ColourInformationBox is present for the media sample or item:

— if the ColourInformationBox describes components by numbers, there shall be at least as many components in the reconstructed image as indicated by the ColourInformationBox, and components are matched by indices;

— if the ColourInformationBox describes components by labels (e.g. Y, Cb / U, Cr / V or R, G, B), these components shall be present in the image after applying palette or filter array, if any.

If the bands need to be identified beyond the component type, it is recommended to use URIs in place of the enumerated component_type value.

5.1.2 Syntax

```
aligned(8) class ComponentDefinitionBox extends Box('cmpd') {
   unsigned int(32) component_count;
   {
      unsigned int(16) component_type;
      if (component_type>=0x8000) {
         utf8string component_type_uri;
      }
   } [component_count];
}
```

5.1.3 Semantics

`component_count` indicates the number of components described in this box.

`component_type` indicates the type of the component, as defined in Table 1.

`component_type_uri` indicates a URI describing the user-defined component type.

5.2 Uncompressed Frame Configuration

5.2.1 Definition

5.2.1.1 Overview

Box Type: `'uncC'`
Container: Video sample entry, `ItemPropertyContainerBox`
Mandatory: Yes, if `codingname` of the sample entry is `'uncv'` or if the `item_type` of the item is `'unci'`
Quantity: One per uncompressed video sample entry or one associated per uncompressed image item

The `UncompressedFrameConfigBox` describes uncompressed frames that are composed of one or more components. RGB or YUV formats are typical examples of such uncompressed frames for which each colour component is a component of the uncompressed frame. Other component types can also be described (e.g. disparity, transparency, infra-red). The type of each component is specified by the `component_type` field in the associated `ComponentDefinitionBox`.

Component values may be absolute values or indexes into a colour palette, with adjustable black and white reference levels. Pattern-based sensor data, such as Bayer, can be described through user-defined patterns, together with various sensor information (polarization, non-uniformity correction, broken pixels, etc.).

For each component, this box specifies the numerical format (e.g. unsigned integer, IEEE 754 binary32 floating point) and bit depth through the `component_format` and `component_bit_depth_minus_one` fields.

Pixel data may be interleaved per component, per pixel, per row or per tile, as specified by the `interleave_type` field, with byte alignment for each row and tile specified by the fields `row_align_size` and `tile_align_size`. Pixel data may also be grouped together in blocks, typically to respect endianness constraints, as specified by the `block_size`, `block_pad_lsb` and `block_little_endian` fields.

Some formats, such as YUV video, do not always use the same 2D resolution for each component of the frame. This is indicated by the `sampling_type` field.

The `UncompressedFrameConfigBox` may indicate a profile, allowing faster identification of the class of video data used. Profiles are defined in subclause 5.3. Some profiles may use version 1 of the `UncompressedFrameConfigBox`, in which case only the profile is given and the `ComponentDefinitionBox`, if absent, is implicitly defined by the profile.

5.2.1.2 Component ordering and constraints

Each pixel in a frame is made of one or more components, where each component is assigned a type (e.g. 'Y', 'U', 'V'), as detailed in subclause 5.1.

The order in which components are specified in the `UncompressedFrameConfigBox` indicates the order in which components are placed into the sample data or within blocks, prior to blocking and endian conversion.

Some formats, such as Bayer image data, contain a 2D array of single-component values, with each individual component value assigned to a component type using a fixed pattern. Such formats are described as mono-component data with no subsampling, use a component type of 11 ('FA'). There shall be at most one component with type 11 ('FA') present in the component list. There may be additional components of other types present, for example to associate an alpha component with a Bayer image. If a component with type 11 ('FA') is present, there shall be a `ComponentPatternDefinitionBox` present in the video sample entry or associated to the item to indicate the pattern of component values.

Some formats code colours according to a set of predefined colours, or palette. Such formats are described as single-component with no subsampling, use a component type of 10 ('P'). There shall be at most one component with type 10 ('P') present in the component list. There may be additional components of other types present, for example to associate per-pixel alpha component with a palette image. If a component with type 10 ('P') is present, there shall be a `ComponentPaletteBox` present in the video sample entry or associated to the item to indicate the palette values.

5.2.1.3 Component size and numerical format

The variable `component_bit_depth` for a component is defined as (`component_bit_depth_minus_one` + 1).

For a given component, the binary representation of each value is given by `component_bit_depth` (the size in bits of each component value) and `component_format`.

The possible values for `component_format` field is defined in Table 2.

Table 2 — Component formats

Value	Description
0	Component value is an unsigned integer coded on `component_bit_depth` bits.
1	Component value is an IEEE 754 binary float number coded on `component_bit_depth` bits (e.g. if `component_bit_depth` is 16, then the component value is coded as IEEE 754 'binary16'). For this component format, `component_bit_depth` values shall be 16, 32, 64, 128 or 256; other values are forbidden.
2	Component value is a complex number coded on `component_bit_depth` bits, where the first `component_bit_depth`/2 bits represent the real part and the next `component_bit_depth`/2 bits represent the imaginary part. Each part is coded as an IEEE 754 binary float of the size `component_bit_depth`/2. For this component format, `component_bit_depth` values shall be 32, 64, 128 or 256; other values are forbidden.
3 to 255	ISO/IEC reserved for future definition.

If `component_align_size` is 0, the component value is coded on `component_bit_depth` bits exactly.

Otherwise (`component_align_size` is not 0), the component value is coded as a word WC of `component_align_size` bytes, starting on a byte boundary. This implies that some pre-alignment padding bits may be present after the previous component value stored; if such pre-alignment padding bits are present, they shall be set to 0. The least significant bit of the component value is located at the least significant bit of WC. Alignment padding bits, if present, are located at the most significant bits and shall be set to 0. If `components_little_endian` is 0, `WC` is stored as a big-endian word. Otherwise (`components_little_endian` is 1), WC is stored as a little-endian word.

NOTE For example, a pixel with 10-bit unaligned (`component_align_size`=0) R, G and B components, followed by a 1-byte aligned 7-bit A component, stored using pixel interleaving mode, would exist in the sample data as 30 consecutive bits containing R, G and B, followed by 2 pre-alignment padding bits for byte alignment, followed by one alignment padding bit then followed by the 7-bit A value (bringing the A value aligned on 1 byte), for a total of 5 bytes.

For each component, `component_align_size` shall be either 0 or such that `component_align_size*8` is greater than `component_bit_depth`. If `components_little_endian` is 1, `block_little_endian` shall be 0 and `component_align_size` of each component shall be different from 0.

Storage of aligned component values is illustrated in Figure 27.

5.2.1.4 Tiling

The uncompressed frame data is structured in a 2D grid of tiles, where the number of tiles in the horizontal direction (resp. vertical direction) is specified by the variable `num_tile_cols_minus_one`+1 (resp. `num_tile_rows_minus_one`+1). Tiles allow grouping together the component values of pixels close to each-other (i.e., in the same spatial region of the frame).

All tiles have the same width and height. The frame width (resp. height) shall be a multiple of `num_tile_cols_minus_one`+1 (resp. `num_tile_rows_minus_one`+1). The tile width is `w/(num_tile_cols_minus_one+1)`, with `w` the frame width, and the tile height is `h/(num_tile_rows_minus_one+1)`, with `h` the frame height.

If the width (resp. height) in a source image is not an integer multiple of `num_tile_cols_minus_one`+1 (resp. `num_tile_rows_minus_one`+1), an application creating the tiled frame shall pad the source image with an appropriate number of columns to the right (resp. rows to the bottom), and the original image dimension shall be documented using a `CleanApertureBox` for media tracks or a clean aperture transformative item property for image items. The width and height fields of the sample entry or the `image_width` and `image_height` fields of the `ImageSpatialExtentsProperty` shall specify the padded width and height.

Tiles shall be stored in raster-scan order: the top-left tile is stored first, followed by the tile to its right, and the first tile of a tile row is stored after the last tile of the previous tile row.

Within a tile, component values shall be stored in raster-scan order, left-to-right and top to bottom: for a given component, the value for pixel {x+1,y} is stored after (but possibly not contiguous with) the value for pixel {x,y}, and the value for pixel {x,y+1} is stored after (but possibly not contiguous with) the value for pixel {x, y}.

5.2.1.5 Sampling mode

5.2.1.5.1 Definition

All components in a frame either have the same dimensions or use pre-defined sampling modes, indicated by the `sampling_type` field. Possible values for this field are described in Table 3.

NOTE 1 A sampling mode can restrict the possible interleaving modes since the number of values is not the same for each component.

NOTE 2 Derived specifications can further restrict the usage of sampling modes with tiling.

Table 3 — Sampling type values

Value	Sampling mode
0	No subsampling
1	YCbCr 4:2:2 subsampling
2	YCbCr 4:2:0 subsampling
3	YCbCr 4:1:1 subsampling
4 to 0xFF	Reserved

5.2.1.5.2 No subsampling

In this mode, all components have the same width and height as the frame.

The tile width and height are not restricted. Derived specifications can further restrict this, for example to enforce width and height to be multiples of 2 in case Y, U and V components are present.

5.2.1.5.3 YCbCr 4:2:2 Subsampling

This sampling mode shall only be used if the following conditions are all true:

— all three `component_type` values 'Y', 'U' and 'V' are present in the component list;

— the tile width is a multiple of 2;

— the `interleave_type` field is set to 0 (component interleaving mode), 2 (mixed interleaving mode) or 5 (multi-Y pixel interleaving mode).

The width and height of the 'Y' component are the width and height of the frame. The height of the 'U' and 'V' component is the same as the height of the 'Y' component. The width of the 'U' and 'V' component is half the width of the 'Y' component. Components of other types may be present, and have the same dimension as the 'Y' component.

Pixels {x,y} and {x+1,y}, with x%2==0, share the same component values 'U' and 'V'.

If `row_align_size` is not 0 and `interleave_type` is 0:

— `row_align_size` shall be a multiple of 2;

— the row alignment for components of type 'U' and 'V', as defined in 5.2.1.7, shall be done using `row_align_size/2`.

If `tile_align_size` is not 0:

— `tile_align_size` shall be a multiple of 2;

— the tile alignment for components of type 'U' and 'V', as defined in 5.2.1.7, shall be done using `tile_align_size/2`.

5.2.1.5.4 YCbCr 4:2:0 Subsampling

This sampling mode shall only be used if the following conditions are all true:

— all three `component_type` values 'Y', 'U' and 'V' are present in the component list;

— both tile width and height are multiple of 2;

— the `interleave_type` field is set to 0 (component interleaving mode) or 2 (mixed interleaving mode).

The width and height of the 'Y' component are the width and height of the frame. The width of the 'U' and 'V' component is half the width of the 'Y' component'. The height of the 'U' and 'V' component is half the height of the 'Y' component. Components of other types may be present, and have the same dimension as the 'Y' component.

Pixels {x,y}, {x+1,y}, {x,y+1} and {x+1,y+1} with x%2==0 and y%2==0, share the same component values 'U' and 'V'.

If `row_align_size` is not 0 and `interleave_type` is 0:

— `row_align_size` shall be a multiple of 2;

— the row alignment for components of type 'U' and 'V', as defined in 5.2.1.7, shall be done using `row_align_size/2`.

If `tile_align_size` is not 0:

— `tile_align_size` shall be a multiple of 4;

— the tile alignment for components of type 'U' and 'V', as defined in 5.2.1.7, shall be done using `tile_align_size/4`.

5.2.1.5.5 YCbCr 4:1:1 Subsampling

This sampling mode shall only be used if the following conditions are all true:

— all three `component_type` values 'Y', 'U' and 'V' are present in the component list;

— tile width is a multiple of 4;

— the `interleave_type` field is set to 0 (component interleaving mode), 2 (mixed interleaving mode) or 5 (Multi-Y pixel interleaving mode).

The width and height of the 'Y' component are the width and height of the frame. The height of the 'U' and 'V' component is the same as the height of the 'Y' component. The width of the 'U' and 'V' component is the width of the 'Y' component divided by 4. Components of other types may be present, and have the same dimension as the 'Y' component.

Pixels {x,y} , {x+1,y} , {x+2,y} , {x+3,y}, with x%4==0, share the same component values 'U' and 'V'.

If `row_align_size` is not 0 and `interleave_type` is 0:

— `row_align_size` shall be a multiple of 4;

— the row alignment for components of type 'U' and 'V', as defined in 5.2.1.7, shall be done using `row_align_size/4`.

If `tile_align_size` is not 0:

— `tile_align_size` shall be a multiple of 4;

— the tile alignment for components of type 'U' and 'V', as defined in 5.2.1.7, shall be done using `tile_align_size/4`.

5.2.1.6 Interleaving modes

5.2.1.6.1 Definition

The interleaving mode of pixels within a tile (or within the frame if a single tile is used) is indicated by `interleave_type`, as defined in Table 4. The `interleave_type` shall apply to all listed components unless stated otherwise in the interleaving mode definition.

NOTE The interleaving describes the layout of values within the sample data. Positioning of sample data within the container file is done according to ISO/IEC 14496-12.

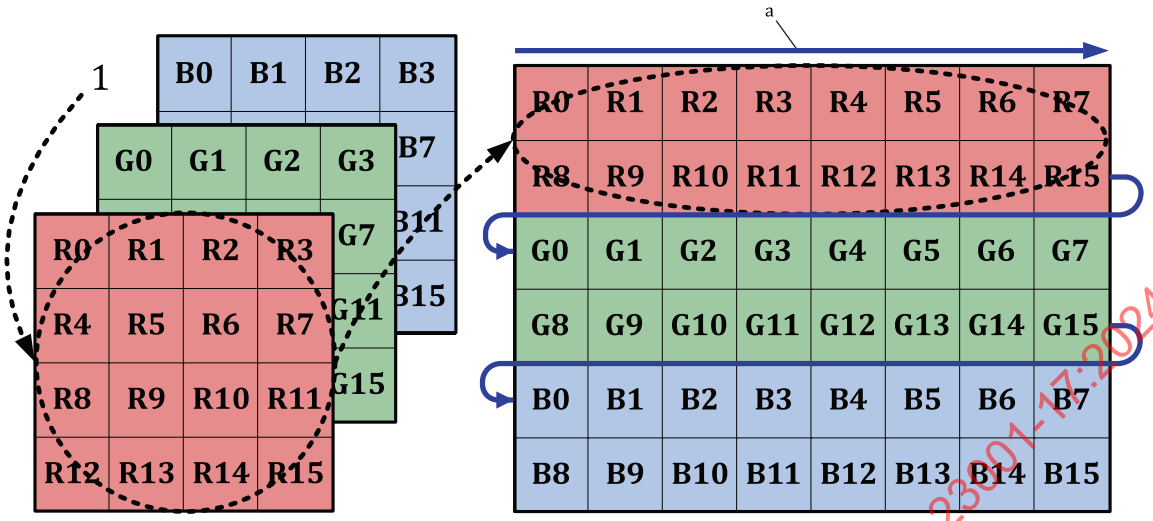
Table 4 — Interleaving type values

Value	Interleaving mode
0	Component interleaving
1	Pixel interleaving
2	Mixed interleaving
3	Row interleaving
4	Tile-component interleaving
5	Multi-Y pixel interleaving
6 to 0xFF	Reserved

5.2.1.6.2 Component interleaving

For a given component, values for all pixels of a tile shall be located sequentially in the sample data. Component values shall be located in the order the components were declared.

Component interleaving mode can be used with a single tile, as illustrated in Figure 1, or with multiple tiles, as illustrated in Figure 2. In both examples, R is the first component, G is the second and B is the third.

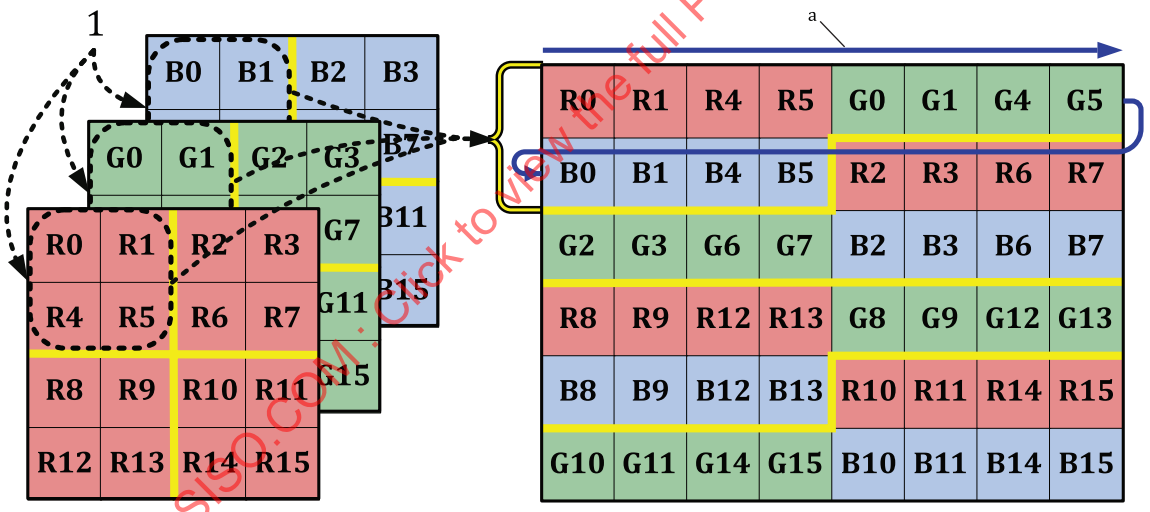


Key

1 frame

a Increasing memory addresses.

Figure 1 — Single tile component interleaving example



Key

1 tile

a Increasing memory addresses.

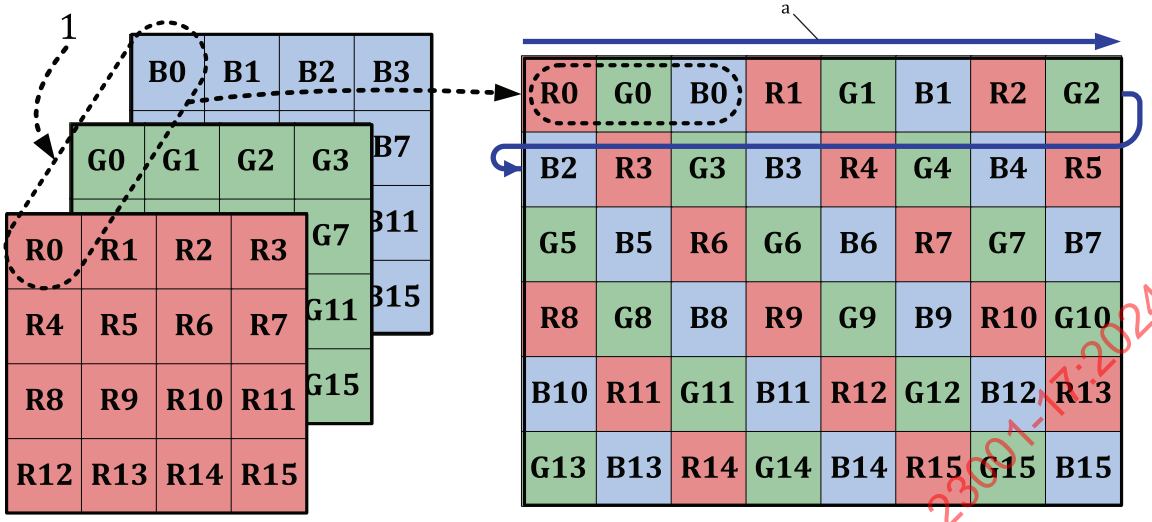
Figure 2 — Multiple tiles component interleaving example

5.2.1.6.3 Pixel interleaving

For a given pixel, the component values shall be located one after the other, in the order the components are declared, with the first component of a pixel following the last component of the previous pixel.

Pixel interleaving mode shall only be used if `sampling_type` field value is 0, i.e. when all components are at the same resolution.

Pixel interleaving mode can be used with a single tile, as illustrated in Figure 3, or with multiple tiles, as illustrated in Figure 4.

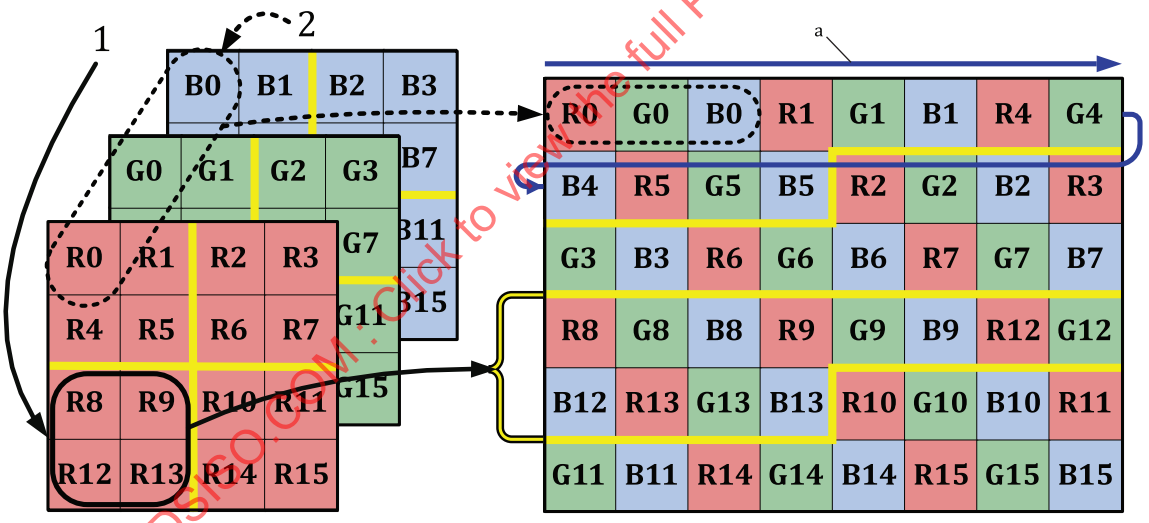


Key

1 pixel

[a] Increasing memory addresses.

Figure 3 — Single tile Pixel interleaving example



Key

1 tile

2 pixel

[a] Increasing memory addresses.

Figure 4 — Multiple tiles Pixel interleaving example

5.2.1.6.4 Mixed interleaving

Mixed interleaving mode shall only be used if the following conditions are all true:

— `sampling_type` field value is not 0;

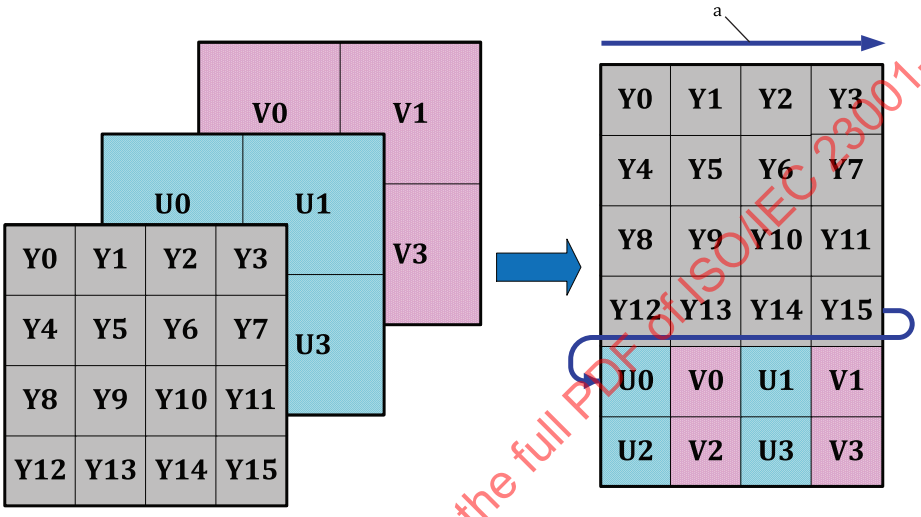
— `sampling_type` value explicitly allows this interleaving mode;

— the ‘U’ and ‘V’ components are consecutive in the component list, i.e. ‘V’ component immediately follows ‘U’ component or ‘U’ component immediately follows ‘V’ component.

For all components other than ‘U’ and ‘V’, component values for all pixels shall be located as specified for component interleaving mode in the order they are declared. The ‘U’ and ‘V’ component values shall be located as specified by pixel interleaving mode; the first value for component ‘U’, if declared first, or ‘V’ otherwise, shall be located after the last value of the component preceding ‘U’ or ‘V’ if any, or first in the tile otherwise.

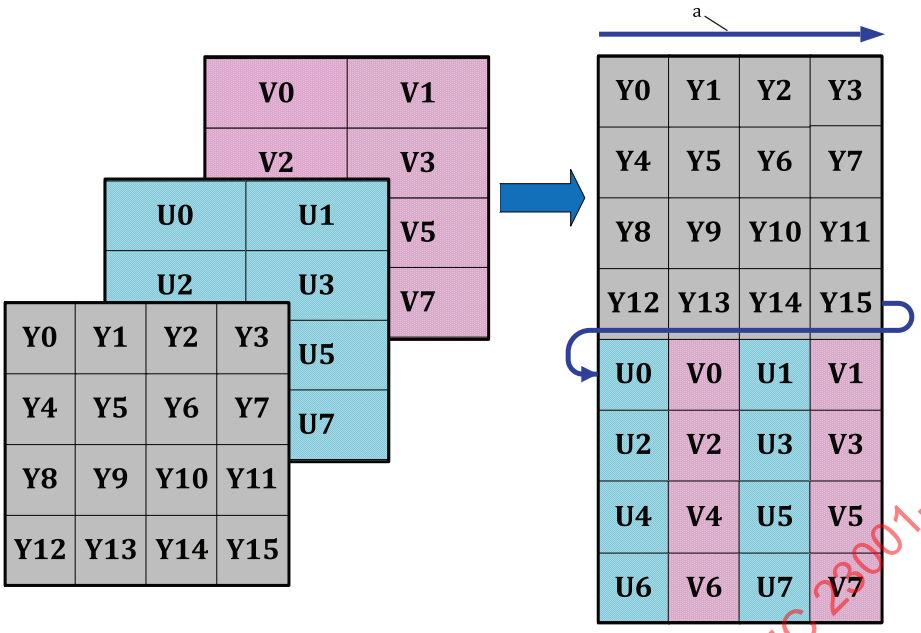
NOTE Mixed interleaving mode is typically used to store YUV 420 data with all Y components followed by interleaved U and V components.

Figure 5 illustrates mixed interleaving for `sampling_type`=2 and Figure 6 illustrates mixed interleaving for `sampling_type`=1.



[a] Increasing memory addresses.

Figure 5 — Mixed interleaving for YUV 420 example



[a] Increasing memory addresses.

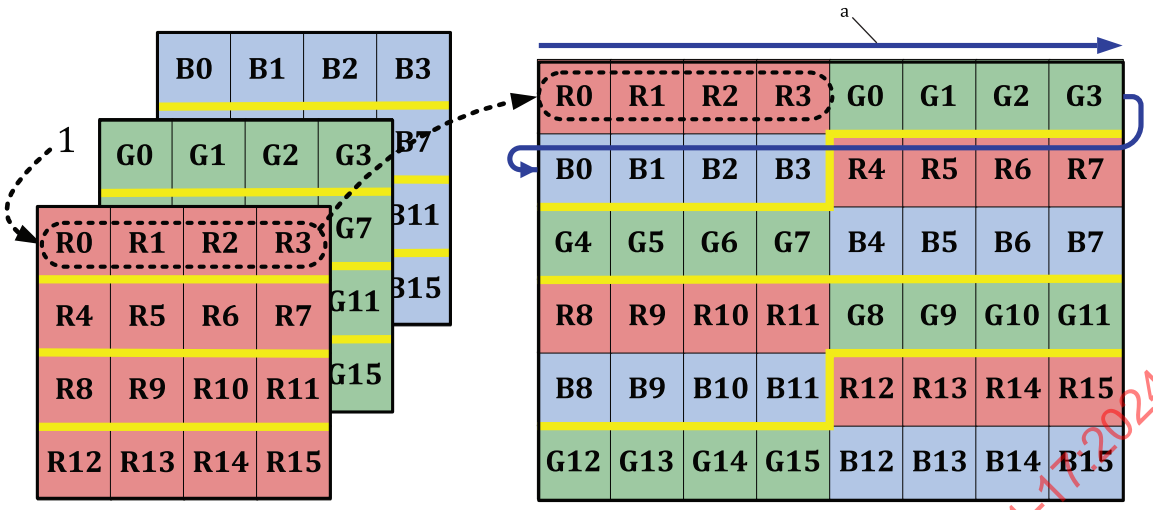
Figure 6 — Mixed interleaving for YUV422 example

5.2.1.6.5 Row interleaving

For a given row, component values for each component shall be located sequentially, in the order the components are declared. Each non-first row in the tile shall be located after the previous row. If multiple tiles are used, the first component of the first row of a tile shall be located after the last component of the last row of the previous tile.

Row interleaving mode shall only be used if `sampling_type` field value is 0, i.e. when all components are at the same resolution.

Row interleaving mode can be used with a single tile, as illustrated in Figure 7, or with multiple tiles, as illustrated in Figure 8.

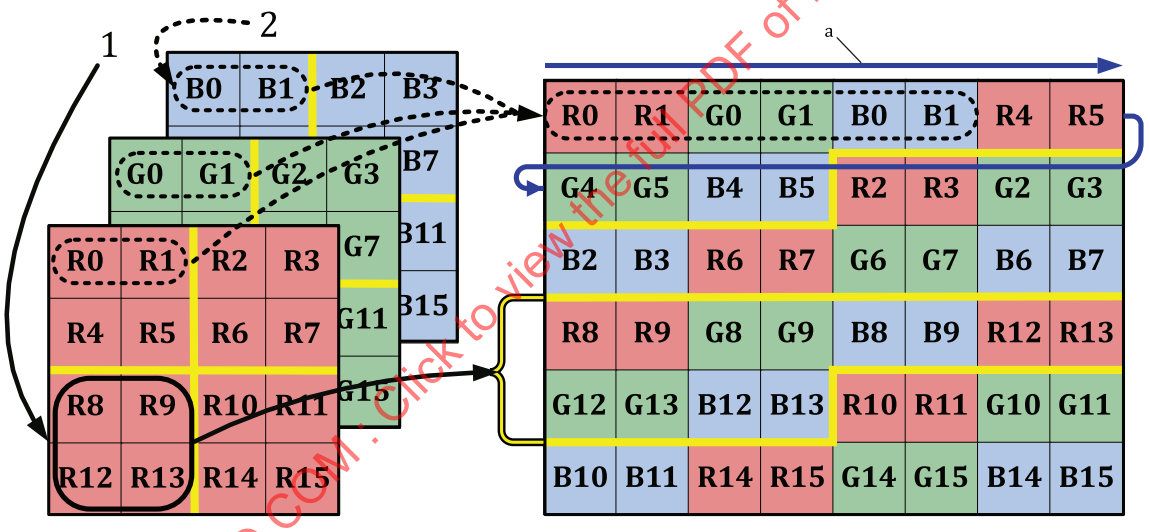


Key

1 row

[a] Increasing memory addresses.

Figure 7 — Single tile row interleaving example



Key

1 tile

2 row

[a] Increasing memory addresses.

Figure 8 — Multiple tiles row interleaving example

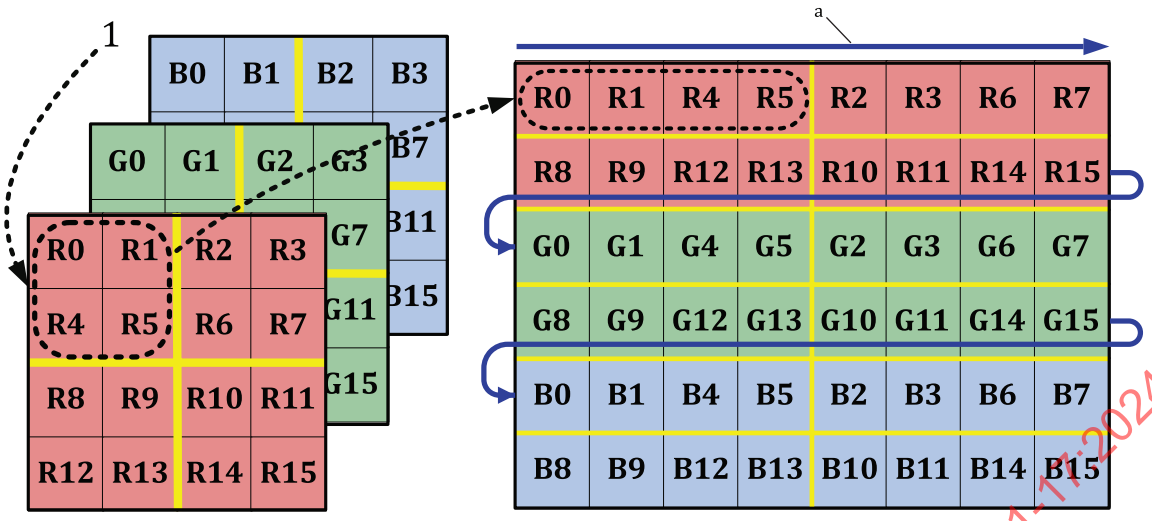
5.2.1.6.6 Tile-component interleaving

For each component, in the order they are declared, the values for each component for all tiles shall be located sequentially, with values within the tile located per each individual tile.

Tile-component interleaving mode shall not be used if only a single tile is defined.

Tile-component interleaving mode shall only be used if `sampling_type` field value is 0, i.e. when all components are at the same resolution.

Tile-component interleaving mode is illustrated in Figure 9.



Key

1 tile

[a] Increasing memory addresses.

Figure 9 — Tile-component interleaving example

5.2.1.6.7 Multi-Y pixel interleaving

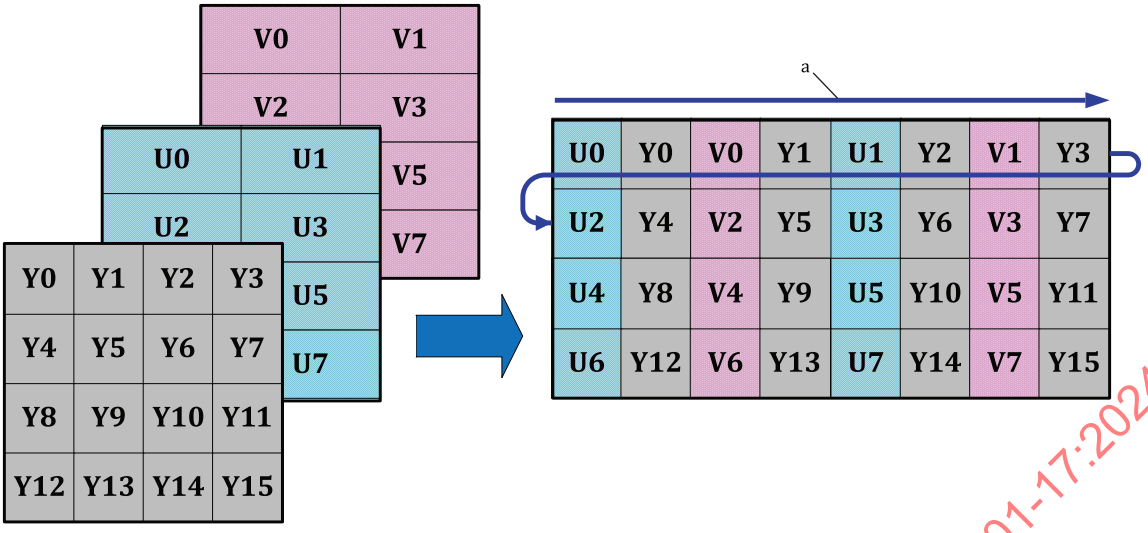
Component values shall be organized as with `interleave_type`=1 (pixel interleaving mode), and the list of components shall contain multiple 'Y' components representing the multiple 'Y' component values associated with a single pair of 'U' and 'V' component values.

The 'U' and 'V' components shall only be listed once in the list of components.

NOTE There are no restrictions on the order of declaration of the 'Y' components; derived specifications can further restrict this.

Regardless of how the multiple 'Y' components are interleaved with the 'U' and 'V' components, the 'Y' component values shall be stored in left-to-right, top-to-bottom order.

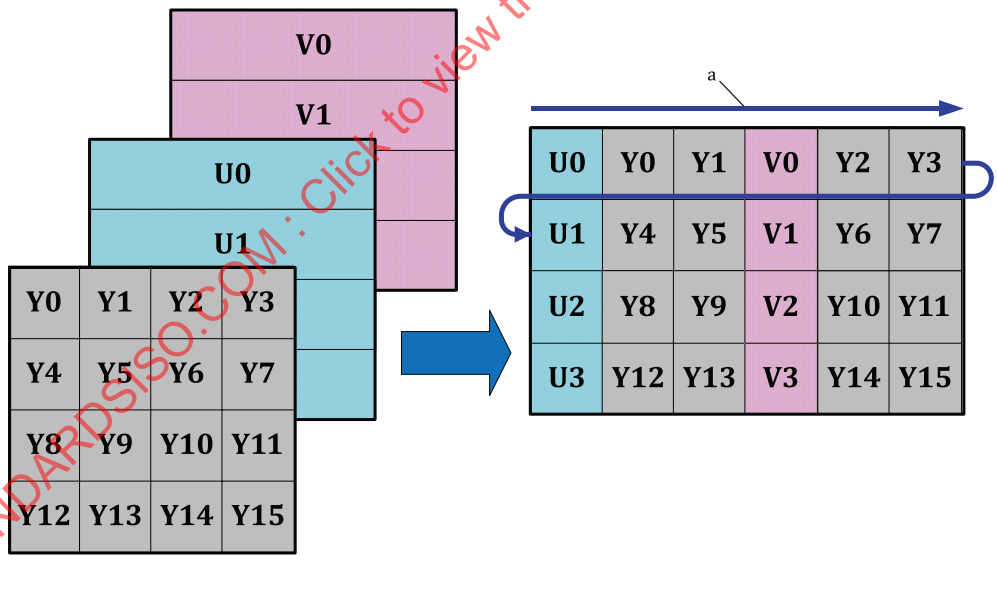
If the chrominance components are subsampled according to 4:2:2 subsampling (`sampling_type`=1), the component list describes two horizontally consecutive pixels, and thus shall include two 'Y' components, representing the two 'Y' values associated with a single pair of 'U' and 'V' component values. For example, {'U', 'Y', 'V', 'Y'} for `sampling_type`=1 (YUV 4:2:2) indicates that the Y value of the second pixel shall be located after the 'V' component value (see Figure 10).



[a] Increasing memory addresses.

Figure 10 — 'U', 'Y', 'V', 'Y' 4 :2 :2 example

If the chrominance components are subsampled according to 4:1:1 subsampling, the component list describes four horizontally consecutive pixels, and thus shall include four Y components, representing the four Y values associated with a single pair of U and V component values. For example, {'U', 'Y', 'Y', 'V', 'Y', 'Y'} for `sampling_type`=3 (YUV 4:1:1) indicates that the 'Y' value of the second pixel shall be located after the 'Y' value of the first pixel, that the 'Y' value of the third pixel shall be located after the 'V' component value and that the 'Y' value of the fourth pixel shall be located after the 'Y' value of the third pixel (see Figure 11).



[a] Increasing memory addresses.

Figure 11 — 'U', 'Y', 'Y', 'V', 'Y', 'Y' 4 :1 :1 example

Multi-Y pixel interleaving mode shall not be used for other forms of chrominance subsampling.

5.2.1.7 Block and value alignment to byte boundaries

Padding bits can be included within the sample data to ensure that component values or pixels are aligned to specific byte boundaries. In addition, the end of rows and tiles can be aligned to specific byte boundaries. This may be to ensure individual component values or pixels are CPU addressable (e.g. not split across two 32-bit words), and rows and tiles of images are aligned to key boundaries (e.g. to memory pages or disk drive sectors).

Component values can be stored either directly in the sample data or inside fixed-size blocks. The block size in bytes is specified by the `block_size` field.

If the block size is 0 (blocking is not used within the sample data):

— no padding is performed between values, except potentially:

  — in-between component values if some components use a `component_align_size` different from 0;

  — after the last component value of a pixel if `pixel_size` is set;

  — after the last value in a row if `row_align_size` is set;

  — after the last value in a tile if `tile_align_size` is set;

— `block_pad_lsb`, `block_little_endian` and `block_reversed` shall all be 0;

— values shall be stored most-significant bit first.

Storage of component values with `block_size`=0 is illustrated in Figure 25.

If the block size is not 0:

— each block shall consist in exactly `block_size` bytes and shall contain exactly one or more component values, i.e. the bits representing a component value are always contained within a single block;

— by default, within a block, the first value of the first component of the top-left pixel of the frame shall be stored in the highest order bits of the first block (big-endian “left to right” format). The second value (for next component or next pixel of same component) shall be stored within the next set of bits in the block up until the last component value being stored in the least significant bits of the block;

— if the block size in bits is greater than the size in bits of component values present in the block, padding bits are present. When present, these padding bits shall be located either in the most or in the least significant bits;

— the block size shall be greater than or equal to the maximum value of `component_align_size` for each component.

  If `pad_unknown` value is 1, the padding bits values are not restricted, otherwise the padding bits shall all be set to 0.

  NOTE 1 Padding bits can be more than 8 (one byte). Unrestricted padding bit values is typically used for legacy content with no restrictions on padded values. Derived specification can further restrict usage of `pad_unknown`.

If blocks are used, the block containing the first component of the top-left pixel of the frame shall begin at the first byte of the sample data. Otherwise (blocks are not used), the most significant bits of the first component of the top-left pixel of the frame shall be located at the most significant bits of the first byte of the sample data.

When `block_pad_lsb` is 0, the least significant bit of the last component value in the block shall be located at the least significant bit of the block and, consequently, any padding bits are located in the most significant bits of the block. Otherwise (`block_pad_lsb` is 1), the most significant bit of the first component value in the block shall be located at the most significant bit of the block and, consequently, any padding bits are located in the least significant bits of the block.

The order of storage, in the sample data, of the bytes within the block is determined by the endian storage type. Figure 12 shows a 32-bit block with bits labelled from 0 (least significant bit) to 31 (most significant bit) with the following component assignments:

Component 1: Type 4 (Red) – 8 bits

Component 2: Type 5 (Green) – 8 bits

Component 3: Type 6 (Blue) – 8 bits

Padding bits are located at the least significant bits of the block.

The least significant bit of each component is labelled '0' and the most significant bit '7'.

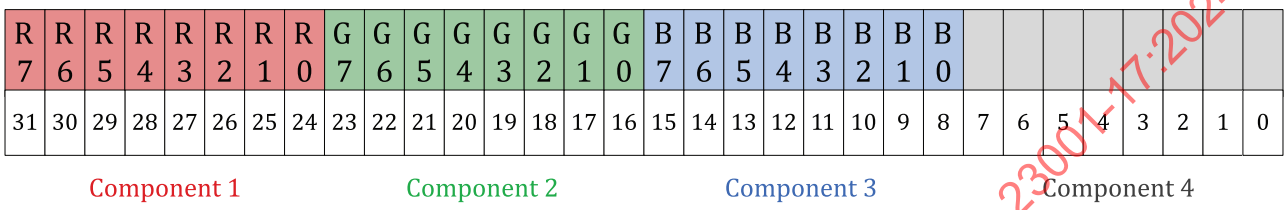


Figure 12 — Assignment of 8-bit RGB components to a 32-bit block

If `block_little_endian` is 0 (big endian storage), the most significant byte of the block (Component 1 in Figure 12) shall be stored in the lowest storage byte location in the sample data. The remaining bytes shall be stored in increasing address spaces (Component 2 then 3 in Figure 12) and the least significant byte of the block (Padding bits in Figure 12) shall be stored in the highest address location. The big-endian storage arrangement for the component assignments of Figure 12 is shown in Figure 13.

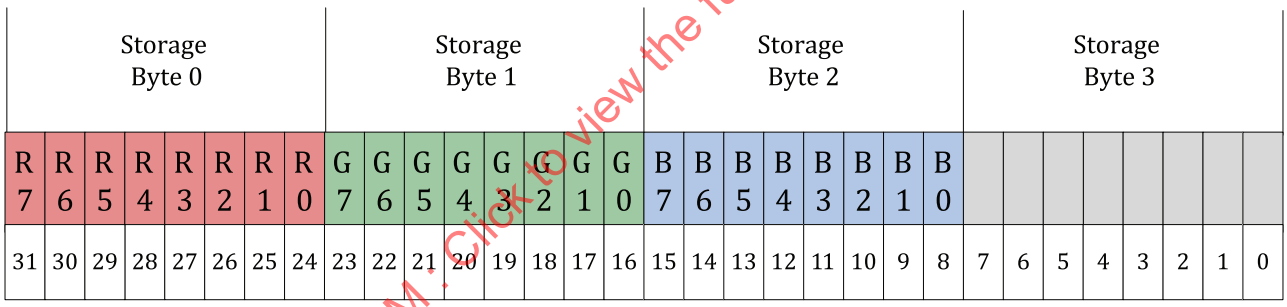


Figure 13 — Big-endian storage of the block defined in Figure 12

If `block_little_endian` is 1 (little endian storage), the least significant byte of the block (Padding bits in Figure 12) shall be stored in the lowest storage byte location. The remaining bytes shall be stored in increasing address space (Component 3 then 2 in Figure 12) and the most significant byte of the block (Component 1 in Figure 12) shall be stored in the highest address location. The little-endian storage arrangement for the block in Figure 12 is shown in Figure 14.

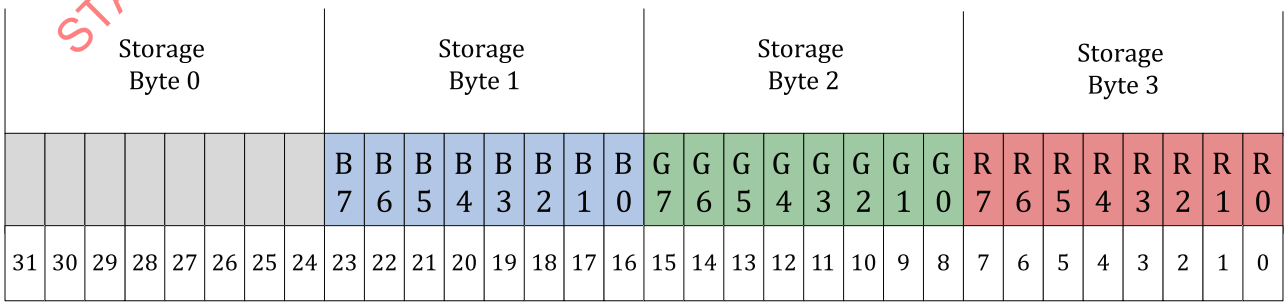


Figure 14 — Little-endian storage of the block defined in Figure 12

If block_reversed is 0, the components order within the block shall be the component declaration order. Otherwise (block_reversed is 1), the partition of components into blocks shall remain unchanged, but the components order within each block shall be inverted, without changing the location of padding bits. block_reversed shall be 0 if block_little_endian is 0.

NOTE 2 Reversing the order of components in a block is typically used with little-endian storage when multiple blocks are needed to store a single pixel value, to ensure the low bits of a block are either a continuation of the previous pixel or a start of a new one.

An example usage of block_reversed=1 is shown in Figure 23 and Figure 24. Figure 29 illustrates usage of block_reversed=1 with multi-Y interleaving mode.

If pixel_size is 0, no additional padding is present after each pixel. Otherwise, let PixelBytes be the number of bytes required to contain all component values of a single pixel, including any padding resulting from block_size or component_align_size. pixel_size shall be greater than or equal to PixelBytes, and the number of padding bytes present after the last byte of each pixel shall be pixel_size - PixelBytes. Usage of pixel_size different from 0 is illustrated in Figure 28.

NOTE 3 pixel_size is typically used to ensure the start of every pixel falls on a significant architectural boundary (e.g. all pixels start on 8-byte boundaries).

pixel_size shall be 0 if interleave_type is different from 1 or 5.

If both block_size and pixel_size are non-zero, then the first component value of any pixel shall be the first value stored in the first block for this pixel. This implies that the last block of any pixel may contain additional padding bits. For example, for a 10-bit RGBA image stored using block_size=4 and pixel_size=10, the first block for each pixel will contain the R, G, and B components and 2 bits of padding while the second block for each pixel will only contain the A component value and 22 bits of padding, and two additional bytes of padding will be present after the second block.

NOTE 4 Specifying pixel_size together with block_size ensures no block contains values from different pixels.

When block_size and pixel_size are both non-zero, the additional padding at the end of each pixel is not required to be a multiple of the block size.

Rows of tiles shall be byte-aligned at the end of the row:

— if the block size is 0, padding bits until byte alignment may be present after the last component value of the last pixel of each row;

— Otherwise, the last block of each row (containing the last component value of the last pixel of each row) may contain more padding bits than previous blocks.

If row_align_size is 0, no additional padding is present at the end of rows of tiles. Otherwise, let RowSize be the number of bytes required to contain, for a given row R:

— all values of all components of row R if interleave_type is 1 or 5 or if interleave_type is 2 and component type is 'U' or 'V' (including all component, block and pixel padding within and at the end of the sample data for row R);

— all values of the current component for row R (including all component and block padding within the sample data for row R) otherwise.

The number of padding bytes present after the last byte of the current row shall be the smallest integer N, with N>=0, such that (RowSize+N) % row_align_size is 0.

If tile_align_size is 0, no additional padding is present at the end of tiles. Otherwise, let TileSize be the number of bytes required to contain, for a given tile T:

— all values of the current component for all rows of the tile T (including all component, block and row padding within and at the end the tile T) if interleave_type is 4;

— all values of all components for all rows of the tile `T` (including all component, block, pixel and row padding within and at the end of the sample data for the tile `T`) otherwise.

The number of padding bytes present after the last byte of the last row of the tile shall be the smallest integer `N`, with `N`>=0, such that `(TileSize+N) % tile_align_size` is 0.

`tile_align_size` shall be 0 if a single tile is used.

NOTE 5 Block alignment allows groups of consecutive component values, regardless of interleaving mode, to be aligned within an easily addressable value; for example, multiple components can be packed into 32-bit values but no component value spans two consecutive 32-bit values. For example, a 10-bit RGBA image using pixel interleaving mode with no component padding and 32-bit (4-byte) block padding on the right will produce a repeated pattern of four 32-bit blocks describing 3 pixels [{R,G,B}, {A, R, G}, {B, A, R}, {G, B, A}]. If tile (resp. row) padding is also specified to be 32-bit (4-byte) then the start the next tile (resp. row) will start at the next 4-byte boundary, regardless of whether space for the next component value was available in the previous block. For example, with 4-byte tile alignment, if the width of the tile is not a multiple of 3, the last block of the line will only contain 1 or 2 component values but all 4-bytes of the last block will be present in the sample data.

5.2.2 Syntax

```
aligned(8) class UncompressedFrameConfigBox extends FullBox('uncC', version, 0) {

unsigned int(32) profile;
if (version==1) {
}
else if (version==0) {
   unsigned int(32) component_count;
   {
      unsigned int(16) component_index;
      unsigned int(8) component_bit_depth_minus_one;
      unsigned int(8) component_format;
      unsigned int(8) component_align_size;
   } [component_count];
   unsigned int(8) sampling_type;
   unsigned int(8) interleave_type;
   unsigned int(8) block_size;
   bit(1) components_little_endian;
   bit(1) block_pad_lsb;
   bit(1) block_little_endian;
   bit(1) block_reversed;
   bit(1) pad_unknown;
   bit(3) reserved = 0;
   unsigned int(32) pixel_size;
   unsigned int(32) row_align_size;
   unsigned int(32) tile_align_size;
   unsigned int(32) num_tile_cols_minus_one;
   unsigned int(32) num_tile_rows_minus_one;
}
}
```

5.2.3 Semantics

`profile` indicates the predefined configuration for this uncompressed frame configuration, as defined in Table 5. Value 0 indicates this uncompressed frame configuration may or may not be compliant to a profile. If not 0, all remaining fields in the box shall have the values indicated in Table 5 for this profile. The four-character code `'gene'` is reserved and shall be interpreted as a value of 0. If version is 1, the profile shall be a valid profile listed in Table 5 for which version=1 is allowed.

`component_count` indicates the number of components present in the frames described by this uncompressed frame configuration.

`component_index` indicates the index of the component listed in the associated `ComponentDefinitionBox`.

`component_bit_depth_minus_one` indicates the size in bits minus one of the values for the component.

`component_format` indicates the format of the component, as defined in Table 2.

`component_align_size` indicates the component byte-alignment size, as described in subclause 5.2.1.3.

`sampling_type` indicates the sampling mode as defined in Table 3.

`interleave_type` indicates the interleaving mode of the components, as defined in Table 4.

`block_size` indicates the size in bytes of blocks, as described in subclause 5.2.1.7.

`components_little_endian` indicates that components are stored as little endian, as described in subclause 5.2.1.3.

`block_pad_lsb` indicates padding bits location, as described in subclause 5.2.1.7.

`block_little_endian` indicates block endianness, as described in subclause 5.2.1.7.

`block_reversed` indicates if component order is reversed within the block, as described in subclause 5.2.1.7.

`pad_unknown` indicates that the value of padded bits in blocks or padded components is unknown.

`pixel_size` indicates the total number of bytes (including component and block padding) used to store all components for a single pixel when component values for that pixel are interleaved, as described in subclause 5.2.1.7.

`row_align_size` indicates the padding between rows, as described in subclause 5.2.1.7.

`tile_align_size` indicates the padding between tiles, as described in subclause 5.2.1.7.

`num_tile_cols_minus_one` plus one indicates the horizontal number of tiles in the frame.

`num_tile_rows_minus_one` plus one indicates the vertical number of tiles in the frame.

5.2.4 Examples

Figures 15 to 29 illustrate the assignment of three components to 32-bit word blocks and associated storage configurations for endian and block padding options. The least significant bit of the block or of each component is labelled '0', the most significant bit is labelled '31'. The three components for most of the examples are defined as follows:

Component 1: Type 4 (Red) – 9 bits

Component 2: Type 5 (Green) – 10 bits

Component 3: Type 6 (Blue) – 9 bits

Figure 15 shows the assignment of the components to a 32-bit block (`block_size`=4) with padding at the most significant bits (`block_pad_lsb`=0).

				R 8	R 7	R 6	R 5	R 4	R 3	R 2	R 1	R 0	G 9	G 8	G 7	G 6	G 5	G 4	G 3	G 2	G 1	G 0	B 8	B 7	B 6	B 5	B 4	B 3	B 2	B 1	B 0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Pad Component 1 Component 2 Component 3

Figure 15 — Example assignment of RGB to 32-bit block, padded at most significant bits

Figure 16 shows the assignment of the components to a 32-bit block (`block_size`=4) with padding at the least significant bits (`block_pad_lsb`=1).

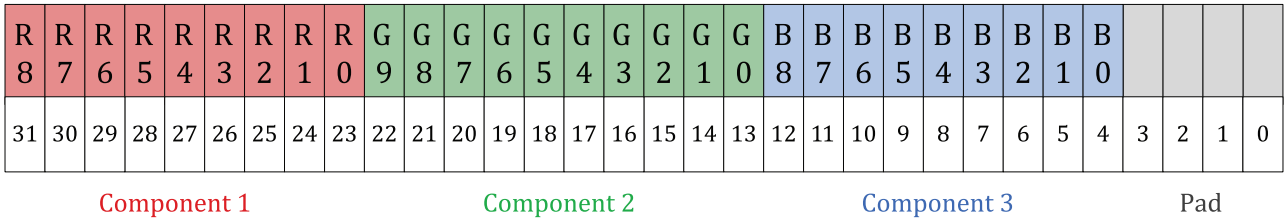


Figure 16 — Example assignment of RGB to 32-bit block, padded at least significant bits

The remaining figures in this section show various combinations of endian, block padding, block reverse, component alignment and pixel alignment options. “Storage Byte 0” (resp. “Storage Byte 3”) in the figures is the first (resp. last) byte stored in the sample data for the illustrated pixel or block.

Figure 17 illustrates a 32-bit block stored in big endian (with `block_little_endian`=0) and padding bits at least significant bits (`block_pad_lsb`=1).

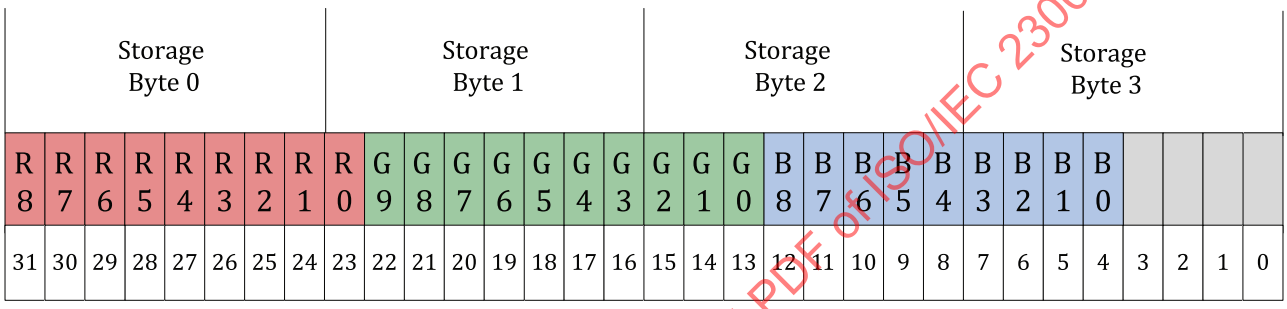


Figure 17 — Big-endian block padded at least significant bits

Figure 18 illustrates a 32-bit block stored in big endian (with `block_little_endian`=0) and padding bits at most significant bits (`block_pad_lsb`=0).

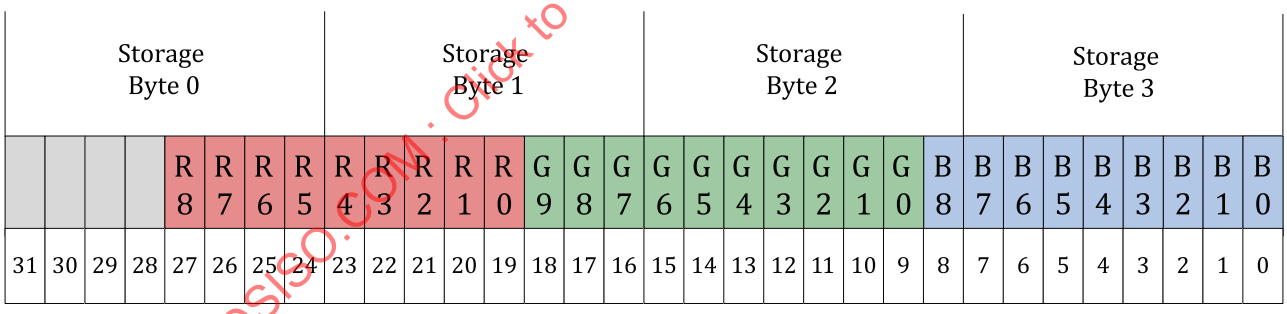


Figure 18 — Big-endian block padded at most significant bits

Figure 19 illustrates a 32-bit block stored in little endian (with `block_little_endian`=1) and padding bits at least significant bits (`block_pad_lsb`=1).

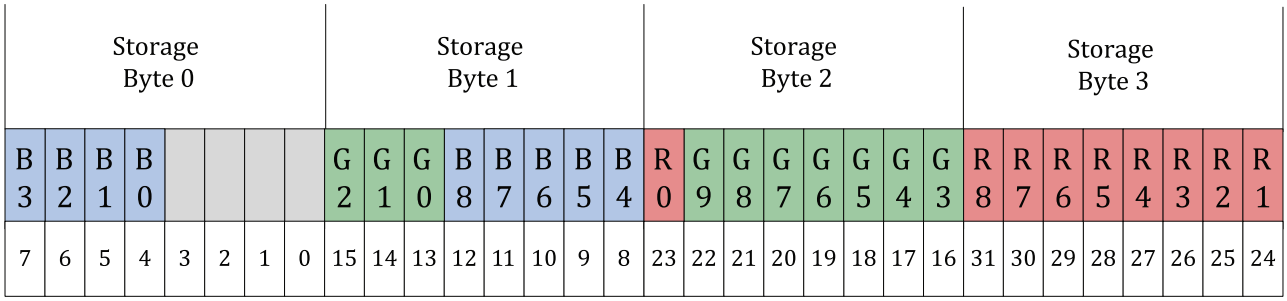


Figure 19 — Little-endian block padded at least significant bit

Figure 20 illustrates Figure 19 with the byte order in the sample data reversed to better show component alignment.

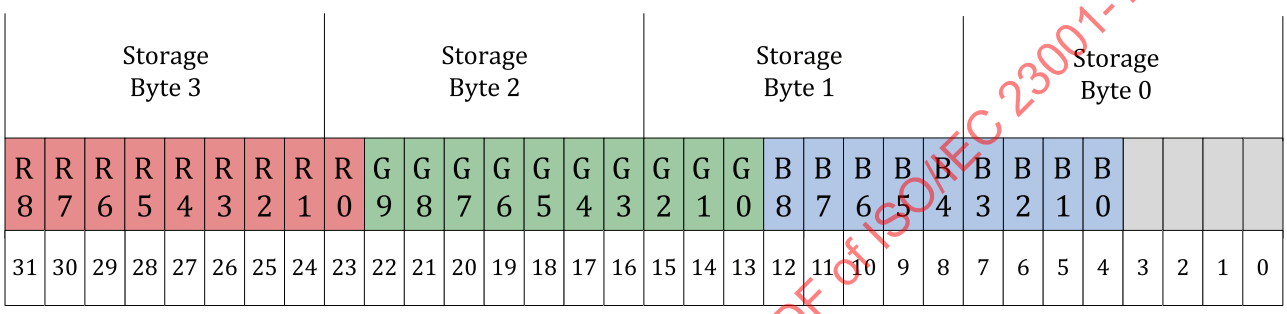


Figure 20 — Little-endian block padded at least significant bit with the storage byte order reversed

Figure 21 illustrates a 32-bit block stored in little endian (with `block_little_endian`=1) and padding bits at most significant bits (`block_pad_lsb`=0).

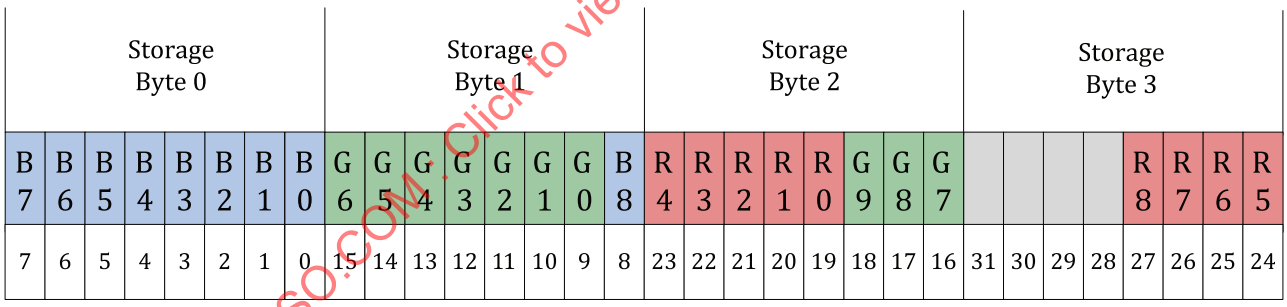


Figure 21 — Little-endian block padded at most significant bit

Figure 22 illustrates Figure 21 with the byte order in the sample data reversed to better show component alignment.

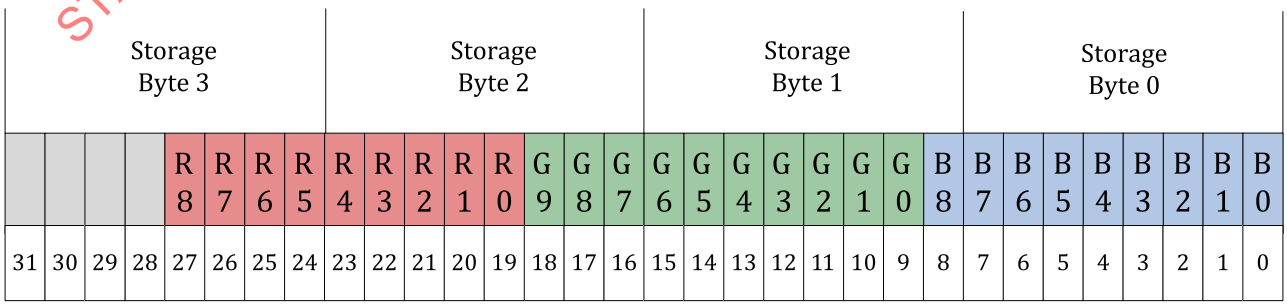


Figure 22 — Little-endian block padded at most significant bit with the storage byte order reversed

Figure 23 illustrates a 32-bit block stored in little endian (with `block_little_endian`=1), padding bits at most significant bits (`block_pad_lsb`=0) and reversed order of component in block (`block_reversed`=1).

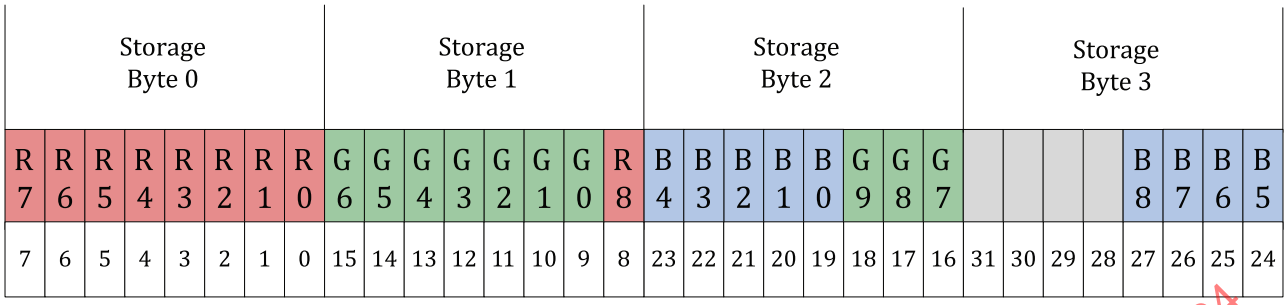


Figure 23 — Little-endian block padded at most significant bit with reversed component order

Figure 24 illustrates Figure 23 with the byte order in the sample data reversed to better show component alignment.

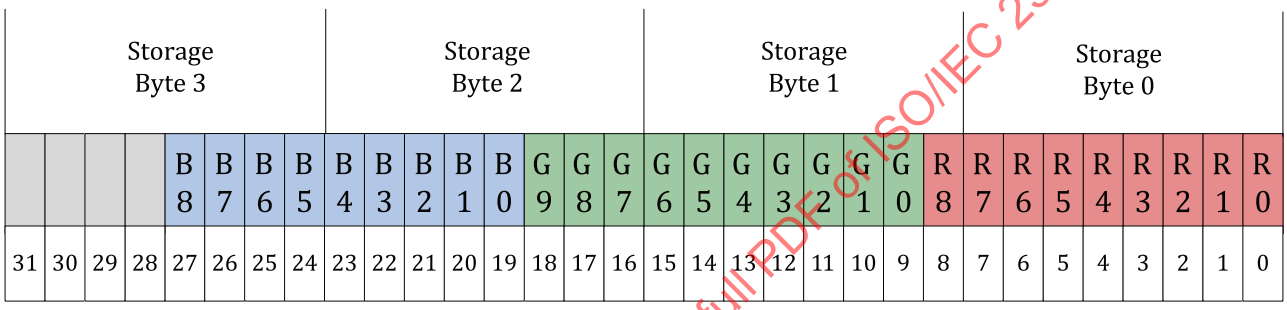


Figure 24 — Little-endian block padded at most significant bit with reversed component order with the storage byte order reversed

Figure 25 shows storage of component Red (9 bits), Green (10 bits) and Blue (9 bits) with no block and no pixel alignment.

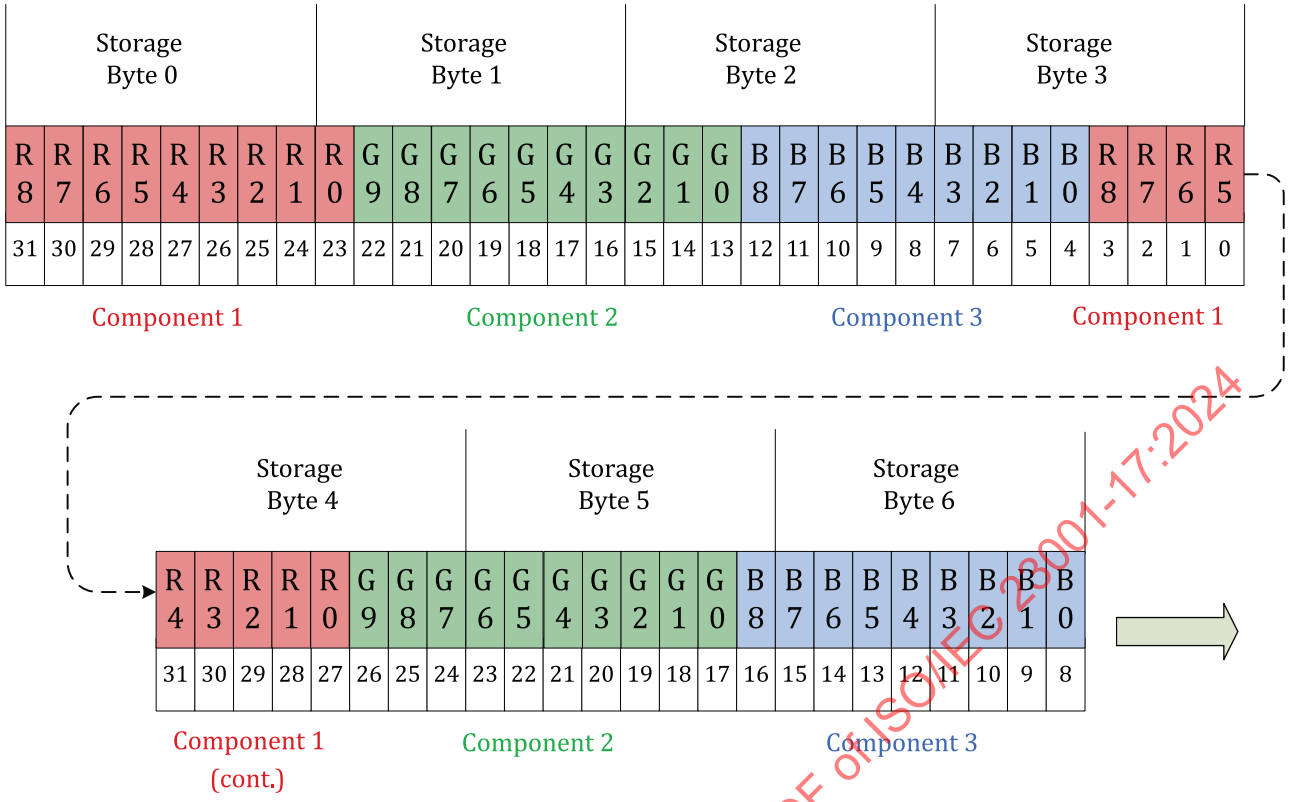


Figure 25 — RGB - red (9 bits), green (10 bits), blue (9 bits), no block, no padding bits

Figure 26 shows storage of components Red, Green, Blue, Alpha, each on 5 bits with no block and 3 bytes per pixels (`pixel_size`=3).

Figure 26 — RGBA (each 5 bits), no block and 3 bytes pixel size

Figure 27 shows storage of components Red (10 bits), Green (12 bits), Blue (10 bits) with no block and `component_align_size`=2 for the Green component only.

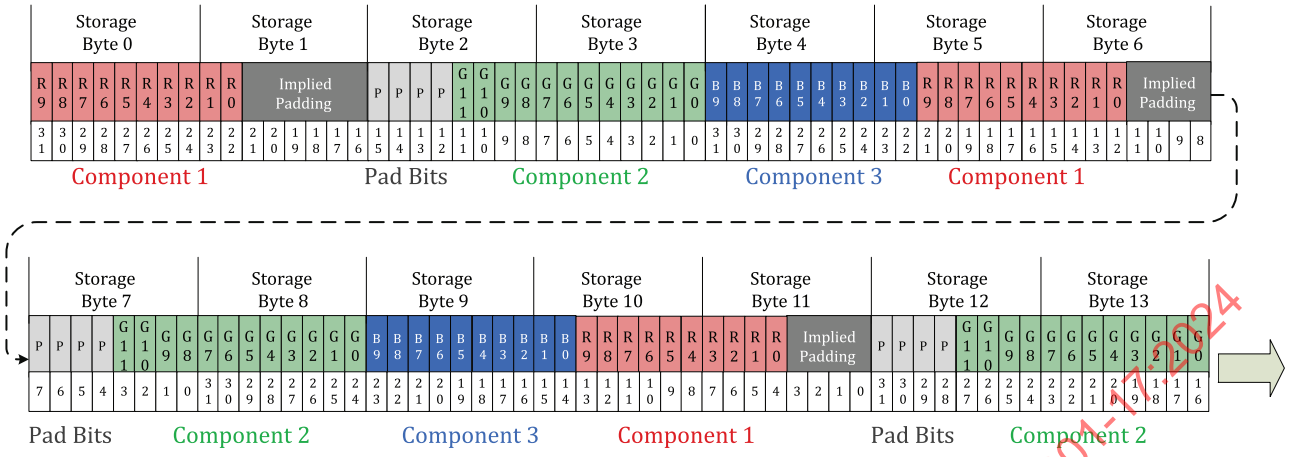


Figure 27 — Red (10-bit unaligned), Green (12-bit, 2-byte alignment), Blue (10-bit unaligned), no block

Figure 28 shows storage of components Red, Green, Blue, Alpha, each on 9 bits within a 32-bit blocks.

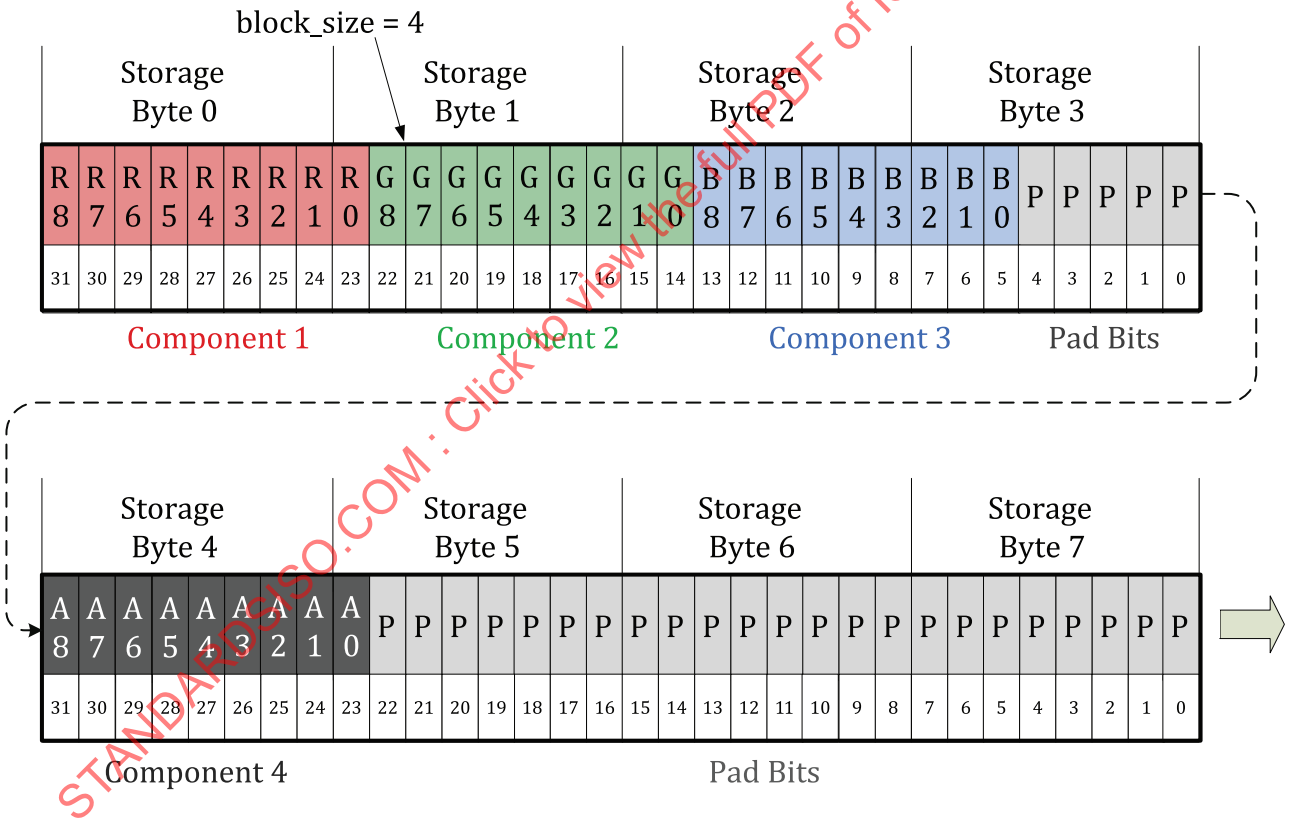


Figure 28 — RGBA (each 9 bits), block size (4 bytes), pixel size (8 bytes)

Figure 29 shows storage of multi-Y interleaved YUV with 422 subsampling, each component coded on 10 bits, within 32-bit blocks with `block_little_endian`=1 and `block_reversed`=1, also known as 'v210' format.

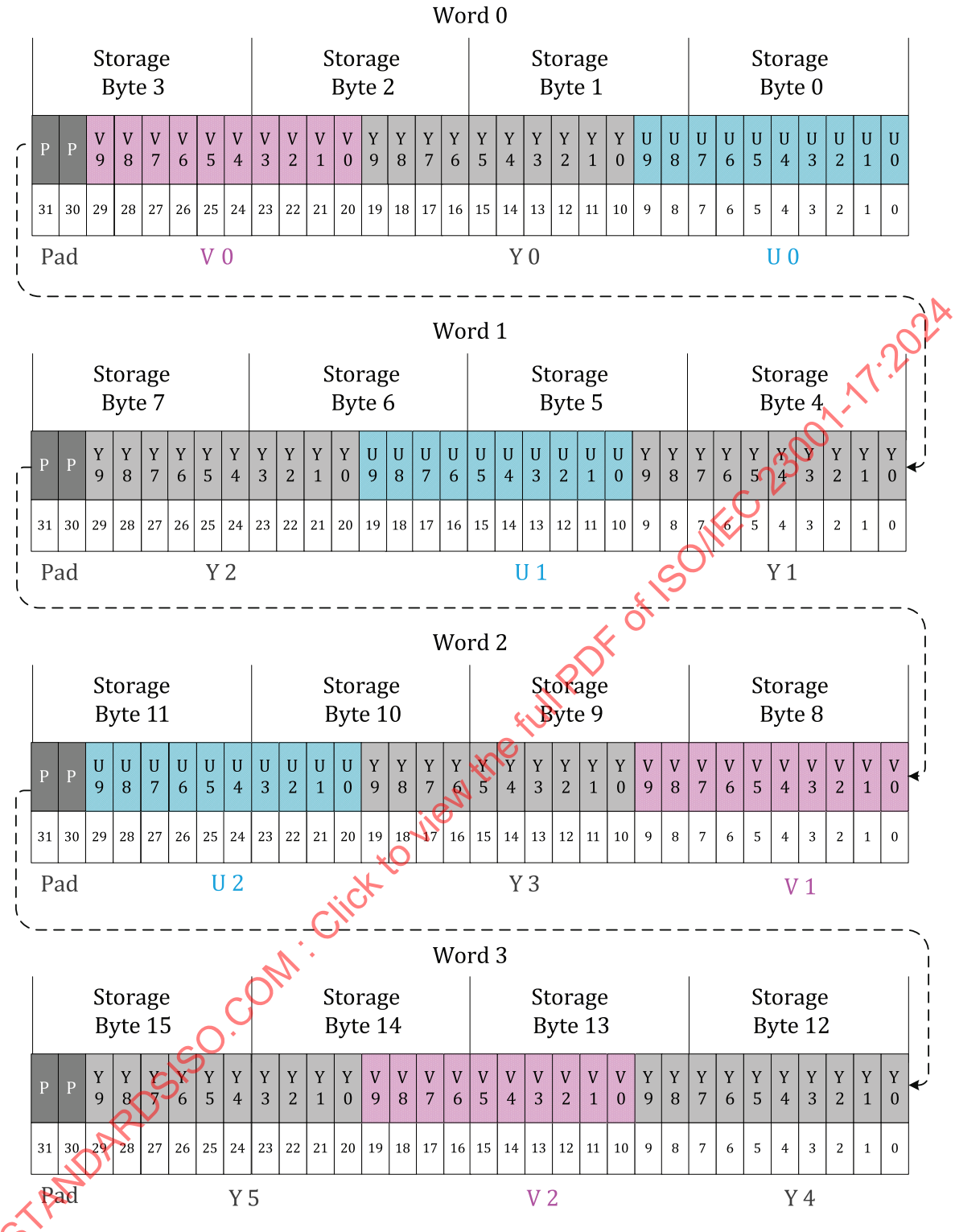


Figure 29 — Multi-Y interleaved component 10-bit UYVY in 32 bit little endian reversed blocks with bytes shown in reverse order per 32-bit word

5.3 Profiles for uncompressed frame configurations

5.3.1 Overview

The four-character codes defined in this subclause identify common formats for which a profile is defined. This profile may be indicated in the `UncompressedFrameConfigBox` to simplify pixel format detection by file processors.

Profiles only provide identification of the pixel format used in uncompressed frames. Image reconstruction may need additional information, such as colour information or chroma location, present in child boxes of the sample entry of the track or associated with the image.

NOTE Some of these profiles are introduced to document existing practices in the industry prior to the definition of this document.

5.3.2 Predefined configurations

In Table 5 below, the values from the `ComponentDefinitionBox` and the `UncompressedFrameConfigBox` are listed as:

{*profile*, [{*component_type*, *component_bit_depth_minus_1*}], *sampling_type*, *interleave_type*}

The array of {component_type, *component_bit_depth_minus_1*} is given in component order as specified in the `UncompressedFrameConfigBox`, and the number of elements in the array gives the value for `component_count`.

Unless indicated otherwise, all other fields of `UncompressedFrameConfigBox`, including flags and version, shall have a value of 0.

If version 1 is allowed for a profile, the implicit `ComponentDefinitionBox` is defined by the array of *component_type* in their order of appearance. For example, for the field values {`'rgb3'`, [{4,7},{5,7},{6,7}], 0, 1}, the implicit `ComponentDefinitionBox` has 3 components [4,5,6].

Table 5 — Predefined uncompressed frame formats

Profile identifier	Description	Field values for `UncompressedFrameConfigBox`
`'2vuy'`	8 bits YUV 422 packed Cb Y0 Cr Y1	{`'2vuy'`, [{2,7},{1,7},{3,7},{1,7}], 1, 5}
`'yuv2'`	8 bits YUV 422 packed Y0 Cb Y1 Cr	{`'yuv2'`, [{1,7},{2,7},{1,7},{3,7}], 1, 5}
`'yvyu'`	8 bits YUV 422 packed Y0 Cr Y1 Cb	{`'yvyu'`, [{1,7},{3,7},{1,7},{2,7}], 1, 5}
`'vyuy'`	8 bits YUV 422 packed Cr Y0 Cb Y1	{`'vyuy'`, [{3,7},{1,7},{2,7},{1,7}], 1, 5}
`'yuv1'`	8 bits YUV 411 packed Y0 Y1 Cb Y2 Y3 Cr	{`'yuv1'`, [{1,7},{1,7},{2,7},{1,7},{1,7},{3,7}], 3, 5}
`'v308'`	8 bits YUV 444 packed Cr Y Cb	{`'v308'`, [{3,7},{1,7},{2,7}], 0, 1}
`'v408'`	8 bits YUVA 444 packed Cb Y Cr A	{`'v408'`, [{2,7},{1,7},{3,7},{7, 7}], 0, 1}
`'y210'`	10 bits YUV 422 packed LE Y0 Cb Y1 Cr	{`'y210'`, [{1,9},{2,9},{1,9},{3,9}], 1, 5} `block_size` shall be 2. `block_little_endian` shall be 1. `block_pad_lsb` shall be 1.
`'v410'`	10 bits YUV 444 packed CbYCr, 2 unused bits	{`'v410'`, [{2,9},{1,9},{3,9}], 0, 1} `block_size` shall be 4. `block_little_endian` shall be 1. `block_pad_lsb` shall be 1. `block_reversed` shall be 1.
`'v210'`	YUV 422 10 bits packed CbYCr	{`'v210'`, [{2,9},{1,9},{3,9},{1,9}], 1, 5} `block_size` shall be 4. `block_little_endian` shall be 1. `block_pad_lsb` shall be 0. `block_reversed` shall be 1. Frame width shall be a multiple of 48. `row_align_size` shall be 0.
`'rgb3'`	RGB 24 bits packed	{`'rgb3'`, [{4,7},{5,7},{6,7}], 0, 1} Version shall be 0 or 1.
`'i420'`	YUV 420 8 bits planar YCbCr	{`'i420'`, [{1,7},{2,7},{3,7}], 2, 0}

Table 5 *(continued)*

Profile identifier	Description	Field values for `UncompressedFrameConfigBox`
`'nv12'`	YUV 420 8 bits semi-planar YCbCr	{`'nv12'`, [{1,7},{2,7},{3,7}], 2, 2}
`'nv21'`	YUV 420 8 bits semi-planar YCrCb	{`'nv21'`, [{1,7},{3,7},{2,7}], 2, 2}
`'rgba'`	RGBA 32bits packed	{`'rgba'`, [{4,7},{5,7},{6,7},{7,7}], 0, 1} Version shall be 0 or 1.
`'abgr'`	RGBA 32bits packed	{`'abgr'`, [{7,7}, {6,7},{5,7},{4,7}], 0, 1} Version shall be 0 or 1.
`'yu22'`	YUV 422 8 bits planar YCbCr	{`'yu22'`, [{1,7},{2,7},{3,7}], 1, 0}
`'yv22'`	YUV 422 8 bits planar YCrCb	{`'yv22'`, [{1,7},{3,7},{2,7}], 1, 0}
`'yv20'`	YUV 420 8 bits planar YCrCb	{`'yu22'`, [{1,7},{3,7},{2,7}], 2, 0}

5.4 MIME type sub-parameters

When the 'codecs' parameter of a MIME type is used, as defined in RFC 6381, its value shall be formatted, using field values of the `ComponentDefinitionBox` and the `UncompressedFrameConfigBox`, as follows (each element in the value is separated from the previous one using a dot '.'):

— The first element of the value shall be set to 'uncv' for video tracks and to 'unci' for image items;

— If the `profile` field is not 0, an element equal to the profile four-character code string (case sensitive) is appended;

— Otherwise (the `profile` field value is 0), the string 'gene' (case sensitive) is appended;

— Optionally if `profile` field is not 0, mandatory otherwise:

  — an element equal to the `sampling_type` expressed in hexadecimal is appended;

  — an element equal to the `interleave_type` expressed in hexadecimal is appended;

  — an element equal to the `block_size` expressed in hexadecimal is appended;

  — an element equal to '*cTr*' is appended, with *c* the number of tile columns in hexadecimal and *r* the number of tile rows in hexadecimal;

  — for each component, in order of appearance, listed in the `UncompressedFrameConfigBox`, an element equal to '*aLb*' may be appended, with *a* the hexadecimal value of `component_type` and *b* the hexadecimal value of `component_bit_depth`. Presence of this element is not mandatory, to limit the size of the 'codecs' parameter for images with very high number of components;

  — finally, an element equal to '*Nz*' may be appended, with *z* the number of components in the `UncompressedFrameConfigBox`.

Hexadecimal value shall not be prefixed with '0x', and leading zeros shall be omitted.

EXAMPLE 1 For an uncompressed video with profile `yvyu`: `codecs=uncv.yvyu`

EXAMPLE 2 For an uncompressed video with monochrome and alpha, same sampling (`sampling_type=0`), component interleaving mode (`interleave_type=0`) with each value stored on 10 bits with 2 bytes blocks little-endian left-padded: `codecs=uncv.gene.0.0.2.1T1.0LA.7LA`

EXAMPLE 3 For an uncompressed video with 2 bits alpha, 10 bits R, B, G, same sampling (`sampling_type=0`), pixel interleaving mode (`interleave_type`=1) with no specific block alignment (`block_size=0`) and a tiling of 4 horizontal for 3 vertical tiles: `codecs=uncv.gene.0.1.0.4T3.7L2.4LA.5LA.6LA`

6 Component description extensions

6.1 Extensions for uncompressed video and uncompressed images

6.1.1 Overview

The boxes defined in this subclause can be used to provide further information on one or more components present in the uncompressed frame. Presence of these boxes may be mandated by the presence of specific components in the uncompressed frame (e.g. presence of a palette component mandates the presence of a `ComponentPaletteBox`).

Each of the defined boxes can be:

— added to a video sample entry for media tracks;

— added as properties associated with an image item.

The syntaxes in this section are given for a video sample entry container and the defined boxes therefore extend `Box` (resp. `FullBox`). When used in an `ItemPropertyContainerBox`, the same syntax applies but the defined boxes extend `ItemProperty` (resp. `ItemFullProperty`). The properties defined in the following sections are descriptive properties.

NOTE Most of the boxes defined in Clause 6 can apply to both uncompressed video formats and compressed video formats. Derived specifications can allow usage of these boxes for compressed formats.

6.1.2 Component Palette configuration

6.1.2.1 Definition

Box Type: `'cpal'`
Container: Video sample entry, `ItemPropertyContainerBox`
Mandatory: No
Quantity: Zero or one

The `ComponentPaletteBox` allows describing image data coded through a palette.

This box shall be present if and only if there is an associated `ComponentDefinitionBox` present, in which there is a component defined with a `component_type` value of 10 ('P').

If `ComponentPaletteBox` is used as an item property, it shall be marked as essential.

The component value of each pixel gives the index in the palette, and shall be an integer between 0 and `values_count-1`, with value 0 indicating the first entry in the list of pixel values of the palette.

In one `ComponentPaletteBox`, there shall not be any listed component with:

— the same `component_type` as a component listed in the associated `UncompressedFrameConfigBox`;

— a `component_type` value of 11 ('FA').

6.1.2.2 Syntax

```
aligned(8) class ComponentPaletteBox extends FullBox('cpal', 0, 0) {
   unsigned int(16) palette_components_count;
   {
      unsigned int(32) component_index;
      unsigned int(8) component_bit_depth_minus_one;
      unsigned int(8) component_format;
   } [palette_components_count];

   unsigned int(32) values_count;
   for (i=0; i<values_count; i++) {
```

```
        for (j=0; j<palette_components_count; j++) {
            bit(X) component_value;
            aligned(8) bit(0) unused_bits_zero;
        }
    }
}
```

6.1.2.3 Semantics

`palette_components_count` indicates the number of components described by the palette.

`component_index` indicates the index of the N^{th} component listed in the associated `ComponentDefinitionBox`.

`component_bit_depth_minus_one` indicates the size in bits minus one of values of the N^{th} component.

`component_format` indicates the format of the N^{th} component, as defined in Table 2.

`values_count` indicates the number of pixel values following.

`component_value` indicates the value of the N^{th} component for the given entry. This field is coded on X bits with X = `component_bit_depth_minus_one` + 1 of the N^{th} component.

`unused_bits_zero` represents the number of unused bits following the component value to achieve byte alignment. These unused bits shall be ignored by the file reader and should be set to 0.

6.1.3 Component Pattern Definition

6.1.3.1 Definition

Box Type: `'cpat'`
Container: Video sample entry, `ItemPropertyContainerBox`
Mandatory: No
Quantity: Zero or one

The `ComponentPatternDefinitionBox` allows describing filter array patterns such as Bayer.

This box shall be present if and only if there is an associated `ComponentDefinitionBox` present, in which there is a component defined with a `component_type` value of 11 ('FA').

If `ComponentDefinitionBox` is used as an item property, it shall be marked as essential.

The pattern is used to assign the final component type to each value. The pattern is defined at the top-left pixel of the image, and is repeated to cover the entire image. For a pixel position {x, y} in the image, with {0,0} being the top-left pixel, the coded value for that position represents the component type given by position `{width%pattern_width, height%pattern_height}` in the `ComponentPatternDefinitionBox`, and other component values for that pixel, as defined in the `ComponentPatternDefinitionBox`, are not present.

The tile width (resp. height) shall be a multiple of `pattern_width` (resp. `pattern_height`).

In one `ComponentPatternDefinitionBox`, there shall not be a listed component with:

— the same `component_type` as a component listed in the associated `UncompressedFrameConfigBox`;

— a `component_type` value of 10 ('P').

EXAMPLE 1 A BGGR Bayer image will be represented by a single component 11 ('FA') and a pattern with 2 rows, 2 columns indicating components [6,5,5,4].

EXAMPLE 2 A GRBG Bayer image will be represented by a single component 11 ('FA') and a pattern with 2 rows, 2 columns indicating components [5,4,6,5].

EXAMPLE 3 A BGGR Bayer image with an alpha plane following the Bayer data will be represented by two components 11 ('FA') and 7 ('A'), an `interleave_type` of 0 and a pattern with 2 rows, 2 columns indicating components [6,5,5,4].

EXAMPLE 4 A Red-White-Green-Blue sensor image will be represented by a single component 11 (“FA”) and a pattern with 4 rows, 4 columns indicating the R, G, B and monochrome components order.

6.1.3.2 Syntax

```
aligned(8) class ComponentPatternDefinitionBox extends FullBox('cpat', 0, 0) {
   unsigned int(16) pattern_width;
   unsigned int(16) pattern_height;
   for (i=0; i< pattern_height; i++) {
      for (j=0; j< pattern_width; j++) {
         unsigned int(32) component_index;
         double(32) component_gain;
      }
   }
}
```

6.1.3.3 Semantics

`pattern_width` indicates the width in pixels of the pattern.

`pattern_height` indicates the height in pixels of the pattern.

`component_index` indicates the index of the N^{th} component listed in the associated `ComponentDefinitionBox`.

`component_gain` indicates a gain to be applied to the N^{th} component, such as for white balance correction with Bayer imagery, etc. The value shall be coded as an IEEE 754 “binary32”.

NOTE The gain for each component is determined by a user defined method for balancing the levels of all components. The application of the gain terms is also a user defined implementation. This is commonly achieved by setting the gain value to one for the component with the highest signal level and to a value greater than one for the remaining components, thereby balancing all the components. Values saturate at the maximum value of a component.

6.1.4 Component Reference Level

6.1.4.1 Definition

Box Type: `'clev'`
Container: Video sample entry, `ItemPropertyContainerBox`
Mandatory: No
Quantity: Zero or one

The `ComponentReferenceLevelBox` allows describing the minimum and maximum values for components present in the image data.

When this box is present, there shall be an associated `ComponentDefinitionBox` present.

When this box is absent or a component type is not listed in this box, reference white and black values for this component type are derived from the `ColourInformationBox` present in the sample entry of the track or associated with the image item. If the `ColourInformationBox` is absent, the levels for the desired component type are derived as follows:

— For components of type Y, U or V with 8 bits depth, reference black is 16 and reference white is 235;

— For components of type Y, U or V with 10 bits depth, reference black is 64 and reference white is for 940;

— Otherwise, reference black is 0 and reference white is maximum value for component bit depth.

When `ComponentReferenceLevelBox` and `ColourInformationBox` are both present and document reference levels for the same component types, information from the `ColourInformationBox` shall be used.

NOTE If the `ColourInformationBox` is present with unspecified `matrix_coefficients` and has `full_range_flag` set to 1, full range is assumed.

Reference levels shall be ignored for non-integer component types.

If `clip_range` is set to 1, `black_level` (resp. `white_level`) indicates the minimum (resp. maximum value) for the component; readers shall clip any value less than `black_level` (resp. greater than `white_level`) to `black_level` (resp. `white_level`).

If `clip_range` is set to 0, readers shall transform the value *N* coded on *k* bits (as read from the sample data) to the value `black_level+`N`*(white_level-black_level)/(2`^{k}`-1)` before display or interpretation.

6.1.4.2 Syntax

```
aligned(8) class ComponentReferenceLevelBox extends FullBox('clev', 0, 0)
{
   unsigned int(32) level_count;
   {
      unsigned int(32) component_index;
      unsigned int(1) clip_range;
      bits(7) reserved = 0;
      signed int(32) black_level;
      signed int(32) white_level;
   } [level_count];
}
```

6.1.4.3 Semantics

`level_count` indicates the number of components for which levels are described.

`component_index` indicates the index of the Nth component listed in the associated `ComponentDefinitionBox`.

`clip_range` indicates if the levels indicate a clip range or an affine transformation of the Nth component values.

`black_level` indicates the black level for the Nth component.

`white_level` indicates the white level for the Nth component; this value shall be greater than the `black_level` value.

6.1.5 Polarization Pattern Definition

6.1.5.1 Definition

Box Type: `'splz'`
Container: Video sample entry, `ItemPropertyContainerBox`
Mandatory: No
Quantity: Zero or more

The `PolarizationPatternDefinitionBox` allows describing a filter array pattern for sensors that have polarization patterns implemented on the image sensor. If the sensor also includes a colour or spectral filter, the pattern dimension is not required to match the pattern dimension given in the `ComponentPatternDefinitionBox`.

When this box is present, there shall be an associated `ComponentDefinitionBox` present.

The pattern is used to assign polarization values at the pixel level. The pattern is defined at the top-left pixel of the image, and is repeated to cover the entire image. For a pixel position {x, y} in the image, with {0,0} being the top-left pixel, the polarization value for each pixel is given by position `{width%pattern_width, height%pattern_height}` in the `PolarizationPatternDefinitionBox`.

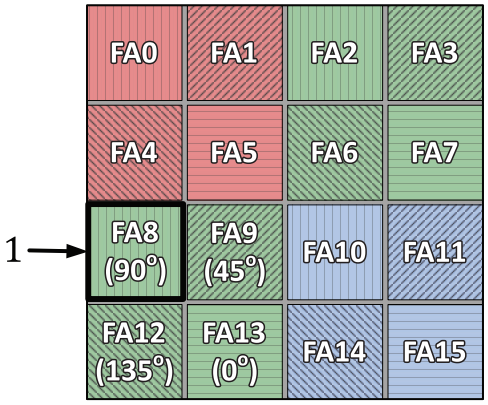
Polarization angles are specified counter-clockwise, with value 0 indicating an orientation to the right, as illustrated in Figure 30.

The tile width (resp. height) shall be a multiple of `pattern_width` (resp. `pattern_height`).

NOTE If both `pattern_width` and `pattern_height` are 1, this implies that a single polarization angle is given for all pixels in the image.

Multiple `PolarizationPatternDefinitionBox` can be specified if components of the image have different polarization filters. In this case, there shall be at most one `PolarizationPatternDefinitionBox` describing one given component of the image.

Figure 30 illustrates an RGGB Bayer filter array pattern superimposed on a 4x4 pixel grid (filter array pattern with values of 'FA1' through 'FA16') with a 2x2 polarization pattern with angles of 90, 45, 135, and 0 degrees.



Key

1 green pixel with 90 degree polarization

Figure 30 — Polarization example with a filter array pattern component

6.1.5.2 Syntax

```
aligned(8) class PolarizationPatternDefinitionBox extends FullBox('splz', 0, 0) {
   unsigned int(32) component_count;
   {
      unsigned int(32) component_index;
   } [component_count]
   unsigned int(16) pattern_width;
   unsigned int(16) pattern_height;
   for (i=0; i< pattern_height; i++) {
      for (j=0; j< pattern_width; j++) {
         double(32) polarization_angle;
      }
   }
}
```

6.1.5.3 Semantics

`component_count` indicates the number of components to which the polarization pattern applies. If this value is 0, the polarization pattern applies to all components of the image.

`component_index` indicates the index of the component listed in the associated `ComponentDefinitionBox`.

`pattern_width` indicates the width in pixels of the pattern.

`pattern_height` indicates the height in pixels of the pattern.

`polarization_angle` indicates the polarization angle of the pixel at the defined pattern location. Value shall either be 0xFFFFFFFF, indicating that no polarization filter is present, or shall be an IEEE 754 "binary32" number, indicating the polarization angle in degrees, with a value greater than or equal to 0.0 and strictly less than 360.0.

6.1.6 Sensor Non-Uniformity Correction

6.1.6.1 Definition

Box Type: `'snuc'`
Container: Video sample entry, `ItemPropertyContainerBox`
Mandatory: No
Quantity: Zero or more

The `SensorNonUniformityCorrectionBox` allows describing pixel specific gain and offset corrections.

When this box is present, there shall be an associated `ComponentDefinitionBox` present.

Whether the correction has applied or not to the values stored in the sample data is indicated by the `nuc_is_applied` field.

The Non-Uniform Correction (NUC) `nuc_gain` and `nuc_offset` attributes provide non-uniformity corrections for a sensor. This is commonly used with radiometric imagery. The correction equation is linear: $y =$ `nuc_gain` $*x+$ `nuc_offset`, where x is an input component value as stored in the file. The minimum value of y saturates at a value of 0. In situations where the same component value range is to be maintained after application of the NUCs, the maximum value of y saturates at a level defined by the number of bits in the component value, for instance, 65535 for a 16-bit component.

The NUC coefficients are assigned per pixel through the full image. The NUC gain and offset tables are aligned with the entire image, resulting in a table width equal to `image_width` and table height equal to `image_height`. For a pixel position {x, y} in the image, with {0,0} being the top-left pixel, the NUC coefficients for each pixel are given by position `{x,y}` in the `SensorNonUniformityCorrectionBox`.

Multiple `SensorNonUniformityCorrectionBox` can be specified if components of the image have different gains and offsets. In this case, there shall be at most one `SensorNonUniformityCorrectionBox` describing one given component of the image.

6.1.6.2 Syntax

```
aligned(8) class SensorNonUniformityCorrectionBox extends FullBox('snuc', 0, 0) {
   unsigned int(32) component_count;
   {
      unsigned int(32) component_index;
   } [component_count]
   unsigned int(1) nuc_is_applied;
   unsigned int(7) reserved=0;
   unsigned int(32) image_width;
   unsigned int(32) image_height;
   for (i=0; i< image_height; i++) {
      for (j=0; j< image_width; j++) {
         double(32) nuc_gain;
      }
   }
   for (i=0; i< image_height; i++) {
      for (j=0; j< image_width; j++) {
         double(32) nuc_offset;
      }
   }
}
```

6.1.6.3 Semantics

`component_count` indicates the number of components to which the non uniformity correction pattern applies. If this value is 0, the non uniformity correction applies to all components of the image.

`component_index` indicates the index of the component listed in the associated `ComponentDefinitionBox`.

`nuc_is_applied` if 1, indicates that the corrections have been applied to the component values in the sample data. If value is 0, the component values are still raw, without NUC corrections applied.