

SMPTE REGISTERED DISCLOSURE DOCUMENT

Avid DNxUncompressed – Packing definition and mapping to the MXF Generic Container



Page 1 of 32

The attached document is a Registered Disclosure Document (RDD) prepared by the proponent identified below. It has been examined by the appropriate SMPTE Technology Committee and is believed to contain adequate information to satisfy the objectives defined in the Scope, and to be technically consistent.

This document is NOT a Standard, Recommended Practice or Engineering Guideline, and does NOT imply a finding or representation of the Society.

Every attempt has been made to ensure that the information contained in this document is accurate. Errors in this document should be reported to the proponent identified below, with a copy to eng@smpte.org.

All other inquiries in respect of this document, including inquiries as to intellectual property requirements that may be attached to use of the disclosed technology, should be addressed to the proponent identified below. At this time the submitter is not holding or aware of others holding any patents, and/or applications for patents, that contain Essential Patent Claims, as defined in the SMPTE Intellectual Property Policy, the use of which would be necessary to implement this document

Proponent contact information:

Avid Technology, Inc.
75 Network Drive
Burlington, MA 01803
Attention: DNx Program Office
Email: dnxcodec@avid.com

Table of Contents	Page
1 Scope	5
2 Conformance Notation	5
3 References	5
4 Terms and Definitions	6
4.1 box.....	6
4.2 byte.....	6
4.3 float16.....	6
4.4 float32.....	6
4.5 fourcc.....	6
4.6 Gbps.....	6
4.7 RGB.....	6
4.8 sizeof(<i>op</i>).....	6
4.9 S2.14	6
4.10 uint8	6
4.11 uint10	7
4.12 uint12	7
4.13 uint16	7
4.14 uint16be	7
4.15 uint32be	7
4.16 uint64be	7
4.17 uint16le	7
4.18 uint32le	7
4.19 uint64le	7
4.20 YC _B CR	7
4.21 '10.6'	7
4.22 '12.4'	7
5 Packing	8
5.1 'pack'	8
5.2 'fill'	9
5.3 Image essence	9
5.3.1 'sinf'	10
5.3.2 'sdat'	14
5.3.3 'icmp'	14
5.4 Image packing by DNxUncompressed profile types	14
5.4.1 'y208'	14
5.4.2 'y408'	14
5.4.3 'y210'	15
5.4.4 'y410'	15
5.4.5 'y212'	16
5.4.6 'y412'	17
5.4.7 'y216', 'y2S1', 'y2p6', 'y2p4'	18
5.4.8 'y416', 'y4S1', 'y4p6', 'y4p4'	18
5.4.9 '\$20y2h'	18
5.4.10 '\$20y2f'	18
5.4.11 '\$20y4h'	19
5.4.12 '\$20y4f'	19
5.4.13 'rg08'	19
5.4.14 'rg10'	19
5.4.15 'rg12'	20
5.4.16 'rg16', 'rgS1', 'rgp6', 'rgp4'	21

5.4.17	'\$20rgh'	21
5.4.18	'\$20rgf'	21
5.4.19	'\$20a08', '\$20y08', 'cb08', 'cr08', '\$20r08', '\$20g08', '\$20b08'	21
5.4.20	'\$20a16', '\$20y16', 'cb16', 'cr16', '\$20r16', '\$20g16', '\$20b16'	22
5.4.21	'\$20y10', 'cb10', 'cr10', '\$20r10', '\$20g10', '\$20b10'	22
5.4.22	'\$20y12', 'cb12', 'cr12', '\$20r12', '\$20g12', '\$20b12'	23
5.4.23	'\$20\$20yh', '\$20cbh', '\$20crh', '\$20\$20rh', '\$20\$20gh', '\$20\$20bh'	24
5.4.24	'\$20\$20yf', '\$20cbf', '\$20crf', '\$20\$20rf', '\$20\$20gf', '\$20\$20bf'	24
5.4.25	'\$20yS1', '\$20yp6', '\$20yp4', 'cbS1', 'cbp4', 'cbp6', 'crS1', 'crp6', 'crp4', '\$20r16', '\$20rS1', '\$20rp6', '\$20rp4', '\$20g16', '\$20gS1', '\$20gp6', '\$20gp4', '\$20b16', '\$20bS1', '\$20bp6', '\$20bp4'	24
5.4.26	'rl08', 'rl16'	24
6	Essence wrapping of DNxPacked	26
6.1	Frame Wrapping (Informative)	26
6.2	Clip Wrapping (Informative)	26
7	KLV Coding of DNxPacked Coding Units	27
7.1	Essence Element Key	27
7.1.1	Essence Element Count – Byte 14	28
7.1.2	Essence Element Type – Byte 15	28
7.1.3	Essence Element Number – Byte 16	28
7.2	Length	28
7.3	Value	28
7.3.1	Frame-wrapped	28
7.3.2	Clip-wrapped	29
8	SMPTE Label for DNxPacked Essence Container Identification	29
9	SMPTE Label for DNxUncompressed Picture Essence Coding	30
10	Essence Descriptors for DNxUncompressed	31
10.1	Generic Picture Essence Descriptor	31
10.2	CDCI Picture Essence Descriptor	31
10.3	RGBA Picture Essence Descriptor	32
11	Application Issues - Alignment	32
12	Bibliography	32

Introduction

This document defines a generic, structured and extensible wrapping format for data (video, audio and data essence) storage in files, as well as various packing methods for video frames wrapped by this format. This is then mapped into a MXF container. Together these elements form the Avid DNxUncompressed format. The format is targeted at overcoming the limitations and pitfalls of administering image sequences in production workflows.

The wrapping format is KLV-based and targeted at byte-based processing and storing on standard computer infrastructure. Serial transmission as a bit-stream is not part of the design consideration. The primary goals are the ability to parse the essence structure with minimal coding investment and the ability to introduce new wrapped elements without breaking existing readers.

The packing methods wrapped by the format either use simple binary packing methods or RLE-based compression for lossless storage of the video essence. Simple implementation, speed of access and processing on byte-based computer system is given priority over packing efficiency. All of the methods have been used in this or a similar form before for storage of individual image frames in computer files.

DNxUncompressed is a picture-by-picture coding scheme where each picture is entirely independent and can be extracted as an independent entity. However, the coding units can be simply concatenated to form a sequence of compressed pictures.

The mapping of the image wrapper to MXF overcomes the metadata gap encountered when working with image sequences, where consistency of image coding parameters across the sequence cannot be ensured, frames can be lost, information relevant for video processing but irrelevant for images is either not recorded or needs to be kept in separately with the inherent risk of loss or separation.

This document maps the DNxUncompressed coding units as either frame-wrapped, where each DNxUncompressed frame is individually mapped into a single MXF Generic Container (GC) Picture Item, or clip-wrapped, where a sequence of DNxUncompressed frames is mapped into a single MXF GC Picture Item. It also defines the KLV coding, the Essence Container and Compression Label values and the Essence Descriptor.

1 Scope

This SMPTE Registered Disclosure Document (RDD) specifies a hierarchical packing method for essence data in general (Part 1), as well as a method for the lossless wrapping of image essence using this packing method in MXF files (Part 2). It specifies the mapping of DNxUncompressed Coding Units as a Picture Element that can be used in the Picture Item of the MXF GC. The Picture Element can contain either individual DNxUncompressed frames using frame-wrapping or a sequence of DNxUncompressed frames using clip-wrapping. The MXF Constrained Generic Container (GC) is described in SMPTE ST 379-2.

2 Conformance Notation

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any section explicitly labeled as "Informative" or individual paragraphs that start with "Note:"

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword "reserved" indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword "forbidden" indicates "reserved" and in addition indicates that the provision will never be defined in the future.

A conformant implementation according to this document is one that includes all mandatory provisions ("shall") and, if implemented, all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

3 References

SMPTE ST 377-1:2011, Material Exchange Format (MXF) – File Format Specification

SMPTE ST 377-1:2011 Am1:2012, Material Exchange Format (MXF) – File Format Specification – Amendment 1

SMPTE ST 377-1:2011 Am2:2012, Material Exchange Format (MXF) – File Format Specification – Amendment 2

SMPTE ST 379-2:2010, Material Exchange Format (MXF) – MXF Constrained Generic Container

SMPTE ST 400:2012, SMPTE Labels Structure

IEEE Std 754-2008: IEEE standard for floating point arithmetic

4 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

4.1 box

byte-aligned KLV structure consisting of a fourcc Key, a 32-bit Length and a variable length Value payload

4.2 byte

sequence of 8 bits.

4.3 float16

16-bit (short) floating point, IEEE 754.compliant storage

4.4 float32

32-bit floating point, IEEE 754.compliant storage

4.5 fourcc

32-bit value formed from a sequence of four 8-bit values (aka characters; usually in ASCII coding, but not necessarily), stored in occurrence order: A fourcc 'abcd' would be stored as a sequence `fourcc[4] = {'a', 'b', 'c', 'd'}`, with 'a' residing at the lowest logical memory address. Non-printable characters (e.g. blank ' ' = 20h) are specified by providing their 2 character hexadecimal value, pre-pended by a '\$', e.g. 'xml\$20'.

4.6 Gbps

gigabits per second. The prefix giga (symbol G) is defined in the International System of Units (SI) as a multiplier of 10^9 (1 billion), and therefore 1 gigabit = 10^9 bits = 1000000000 bits.

4.7 RGB

signal storage format using interleaved R, G and B components (see also $YCbCr$). Individual components (non-interleaved) can be stored using planar signal formats. The specifics of the component coding are described in the Picture Essence Descriptors of the MXF container.

4.8 sizeof(*op*)

stored size, in bytes, of the operand *op*.

4.9 S2.14

16-bit signed fixed-point format, allowing 14 bits fractional precision and covering the range $[-2, 2]$. A S2.14 value is stored as a signed 16-bit integer value *v* which is mapped to a float value *f* as $f = ((\text{float})v)/16384$.

4.10 uint8

unsigned 8- bit integer

4.11 uint10

unsigned 10-bit integer

4.12 uint12

unsigned 12-bit integer

4.13 uint16

unsigned 16-bit integer

4.14 uint16be

unsigned 16-bit integer, big-endian

4.15 uint32be

unsigned 32-bit integer, big-endian

4.16 uint64be

unsigned 64-bit integer, big-endian

4.17 uint16le

unsigned 16-bit integer, little-endian

4.18 uint32le

unsigned 32-bit integer, little-endian

4.19 uint64le

unsigned 64-bit integer, little-endian

4.20 YC_BC_R

signal format using an interleaved color difference format (see also RGB). Individual components (non-interleaved) can be stored using planar signal formats. The specifics of the component coding are described in the Picture Essence Descriptors of the MXF container.

4.21 '10.6'

16-bit format storing unsigned 10 bit signal data in the 10 MSBs of the 16-bit value and (optionally) allowing 6 bits for fractional precision. The valid range of values is [0.0, 1023.0], excluding the last 33 values of the 16-bit range (FFC1h-FFFFh). Data are treated and marked as 10 bit.

4.22 '12.4'

16-bit format storing unsigned 12 bit signal data in the 12 MSBs of the 16-bit value and (optionally) allowing 4 bits for fractional precision. The valid range of values is [0.0, 4095.0], excluding the last 15 values of the 16-bit range (FFF1h-FFFFh). Data are treated and marked as 12 bit.

Part 1 – The DNxPacked byte-stream format

5 Packing

The packing format used by DNxUncompressed is a generic byte-stream which consists of KLV box objects. To define these objects this document uses a C++-like pseudo-code notation similar to the ISO base media file format (ISO 14496-12). Elements shall be stored in the order they are defined, byte aligned and close-packed. Derivation implies that all information present in the base class is also present in the derived class, and that, in the bitstream, all such information precedes any additional bitstream syntax declarations specified in the derived class. This generic format is referred to as 'DNxPacked' hereafter.

This document defines the packing types applicable to DNxUncompressed within the DNxPacked structure (see section 5.4).

A box shall consist of a Key-Length `box_header` and a variable-sized payload. The `box_header` shall be defined as

```
box_header(fourcc type)
{
    uint32le    box_size;
    fourcc      box_type = type;
};
```

The `box_size` shall define the outside size of the box, including the header and the payload. The size of a box shall be equal or greater than 8 bytes.

The `box_type` shall define the type of the box and how its payload shall be interpreted.

A *box* object shall be constructed as

```
box(type, payload) : box_header( type )
{
    payload_type box_payload    = payload;
    box_header.box_size        = 8+sizeof(box_payload);
                                // 8: sizeof(box_header)
};
```

The payload shall either be a closely-packed list of 1 to N boxes or directly interpretable fields, but not a mixture of both. A decoder/parser processing a box containing other boxes shall skip any unknown box type and continue processing. To allow future versions of this document to extend the definition of a box without the need to introduce reserved fields, a decoder shall always read the full stored `box_size` and ignore any extra bytes exceeding the size of the payload defined in this document.

5.1 'pack'

```
pack_box: box_header( 'pack' )
{
    box    box_list[];
    box_header.box_size = 8+sum_of(box_list[i].box_size);
};
```


Each essence unit shall be wrapped into a single `pack_box` of type **'pack'**, which shall contain a list of other boxes further describing the essence coding. Essence types shall not be mixed within one `pack_box`.

The `pack_box` shall form the coding unit used by the MXF mapping definition.

5.2 'fill'

```
fill_box(fillerSize) : box_header('fill')
{
    uint8 filler[fillerSize];
    boxheader.box_size = 8+fillerSize;
};
```

Fill boxes shall be skipped during parsing. The payload is undefined and shall not be interpreted. Fill boxes can be used, at the encoders discretion, to align following boxes to specific boundaries.

5.3 Image essence

If the `pack_box` contains image essence it shall contain minimum one `signal_data_box` **'sdat'**, containing the packed image data, and one `signal_info_box` **'sinf'**, describing said image data, as follows:

Each **'sinf'/'sdat'** pair shall be packed into a `img_component_box` **'icmp'**. If only a single pair is present the `img_component_box` may be dropped.

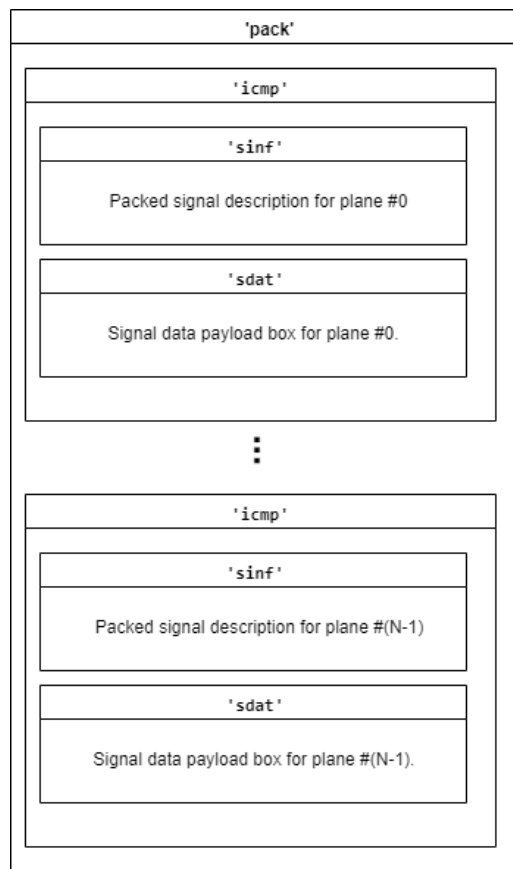


Figure 1 - Packed image essence

5.3.1 'sinf'

```

signal_info_box(width, height, layout, type) : box_header( 'sinf' )
{
    uint32le  frame_width = width;
    uint32le  frame_height= height;
    fourcc     packed_type = type;
    uint8      frame_layout= layout;

    box_header.box_size = 21;
};

```

The `frame_width` and `frame_height` shall describe the sampled region of the coded image. Images shall be stored left-to-right, top-to-bottom. The `packed_type` may define further constraints.

The `frame_layout` shall be 0 for progressive frames (full frame) and 1 for interlaced frames (mixed fields). See SMPTE ST 377-1, Annex G.2.1 for a definition of these layouts.

The `packed_type`, as applicable to DNxUncompressed, is a fourcc profile label from Table 1 or Table 2.

Profile (fourcc)	Sample Format	Sample Type	Subsampling	Gbps (1920 x 1080 @59.94p) [informative]
y208	YCbCr	uint8	4:2:2	1.988
y408			4:4:4	2.982
y210		uint10	4:2:2	2.485
y410			4:4:4	3.728
y212		uint12	4:2:2	2.982
y412			4:4:4	4.474
y216		uint16	4:2:2	3.977
y416			4:4:4	5.965
\$20y2h		float16	4:2:2	3.977
\$20y4h			4:4:4	5.965
\$20y2f		float32	4:2:2	7.954
\$20y4f			4:4:4	11.930
y2S1		S2.14	4:2:2	3.977
y4S1			4:4:4	5.965
y2p6		10.6	4:2:2	3.977
y4p6			4:4:4	5.965
y2p4		12.4	4:2:2	3.977
y4p4			4:4:4	5.965
rg08	RGB	uint8	(4:4:4)	2.982
rg10		uint10		3.728
rg12		uint12		4.474
rg16		uint16		5.695
\$20rgh		float16		5.965
\$20rgf		float32		11.932
rgS1		S2.14		5.965
rgp6		10.6		5.965
rgp4		12.4		5.965

Table 1 – Fourcc codes for DNxUncompressed interleaved signal types

Profile (fourcc)	Sample Format	Sample Type	Gbps (1920 x 1080 @59.94p) [informative]
\$20a08	Alpha/ Mono	uint8	0.994
\$20y08	Y		
cb08	C _B		
cr08	C _R		
\$20r08	R		
\$20g08	G		
\$20b08	B		
\$20a16	Alpha/ Mono	uint16	1.998
\$20y16	Y		
cb16	C _B		
cr16	C _R		
\$20r16	R		
\$20g16	G		
\$20b16	B		
\$20y10	Y	uint10	1.242
cb10	C _B		
cr10	C _R		
\$20r10	R		
\$20g10	G		
\$20b10	B		
\$20y12	Y	uint12	1.491
cb12	C _B		
cr12	C _R		
\$20r12	R		
\$20g12	G		
\$20b12	B		
\$20\$20yh	Y	float16	1.988
\$20cbh	C _B		

\$20crh	C_R		
\$20\$20rh	R		
\$20\$20gh	G		
\$20\$20bh	B		
\$20\$20yf	Y	float32	3.977
\$20cbf	C_B		
\$20crf	C_R		
\$20\$20rf	R		
\$20\$20gf	G		
\$20\$20bf	B		
rl08	Alpha/ Mono	uint8	variable
rl16		uint16	
\$20yS1	Y	S2.14	1.988
cbS1	C_B		
crS1	C_R		
\$20rS1	R		
\$20gS1	G		
\$20bS1	B		
\$20yp6	Y	10.6	1.988
cbp6	C_B		
crp6	C_R		
\$20rp6	R		
\$20gp6	G		
\$20bp6	B		
\$20yp4	Y	12.4	1.988
cbp4	C_B		
crp4	C_R		
\$20rp4	R		
\$20gp4	G		
\$20bp4	B		

Table 2 - Fourcc codes for DNxUncompressed planar signal types

The exact packing details for these types are defined in section 5.4.

5.3.2 'sdat'

```

signal_data_box: box_header( 'sdat' )
{
    uint8 data[];

    box_header.box_size = 8+sizeof(data);
};

```

The `signal_data_box` shall contain the essence, packed as described by the paired `signal_info_box`.

5.3.3 'icmp'

```

img_component_box( type ): box_header( 'icmp' )
{
    signal_info_box  info( type );
    signal_data_box  data;

    box_header.box_size = 8+info.box_size+data.box_size;
}

```

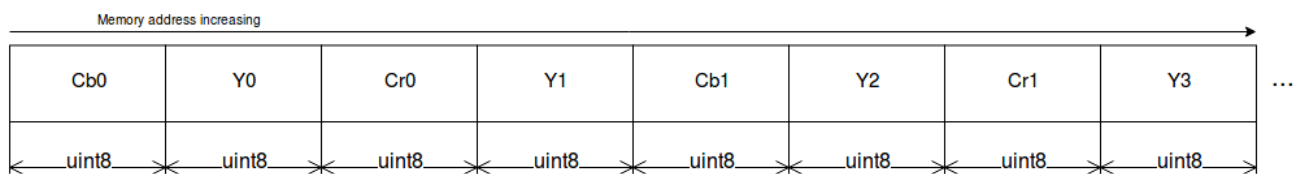
The `img_component_box` shall contain exactly 1 pair of 'sinf'/'sdat' boxes. 'icmp' boxes may appear in any order within a 'pack' box.

5.4 Image packing by DNxUncompressed profile types

5.4.1 'y208'

Signal: YCbCr, 4:2:2, 8 bit, interleaved

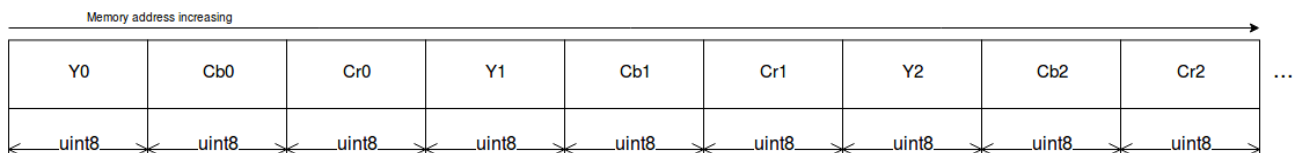
The image shall be packed row by row. Image width must be a multiple of 2. The color components shall be packed in Cb0-Y0-Cr0-Y1 order.



5.4.2 'y408'

Signal: YCbCr, 4:4:4, 8 bit, interleaved

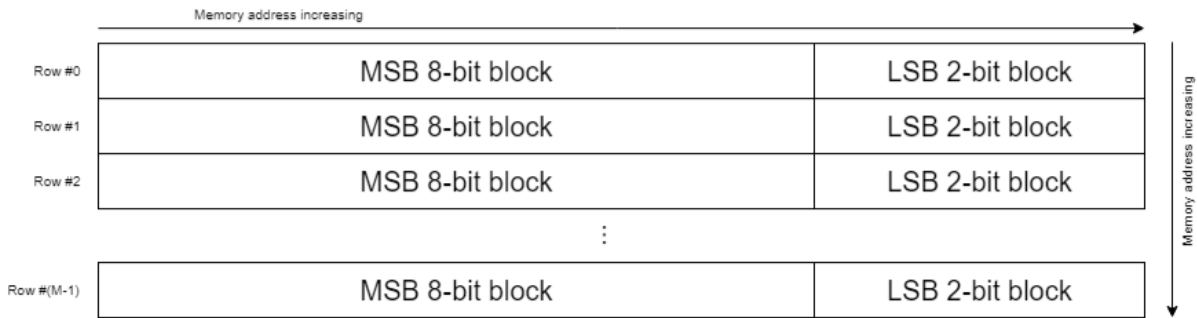
The image shall be packed row by row. The color components shall be packed in Y-Cb-Cr order:



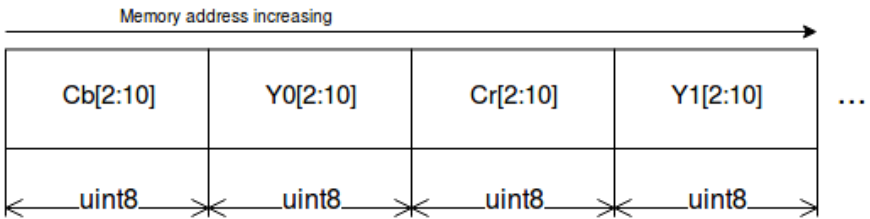
5.4.3 ‘y210’

Signal: YCbCr, 4:2:2, 10 bit, interleaved

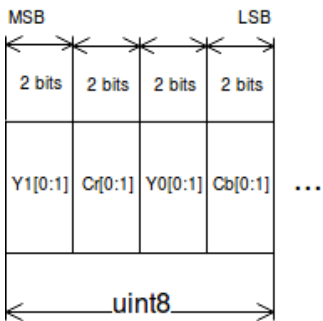
The image shall be packed row by row. Image width must be a multiple of 2. The 8 MSB bits of all the color components of a row shall be stored in 1 block first, then the 2 LSB bits of all the color components of the row shall follow in a second block.



The 8 MSB bits shall be stored in the order Cb-Y0-Cr-Y1.



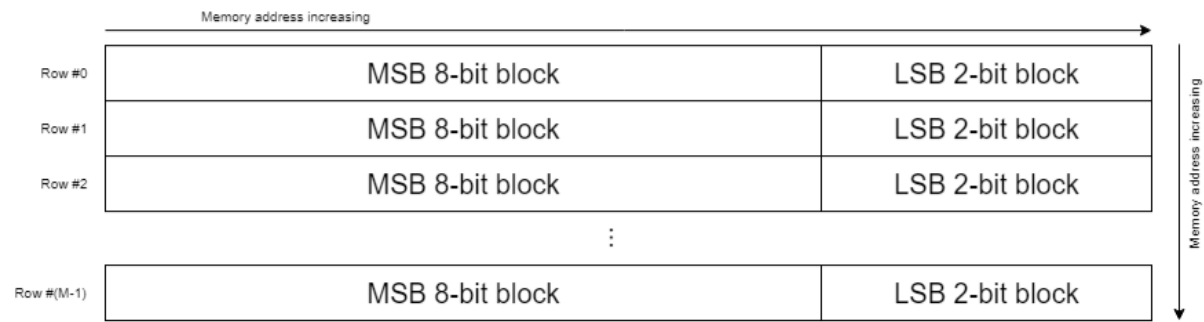
The 2 LSB bits shall also be stored in the order Cb-Y0-Cr-Y1:



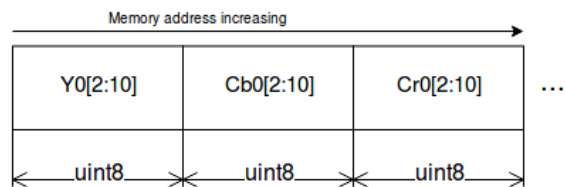
5.4.4 ‘y410’

Signal: YCbCr, 4:4:4, 10 bit, interleaved

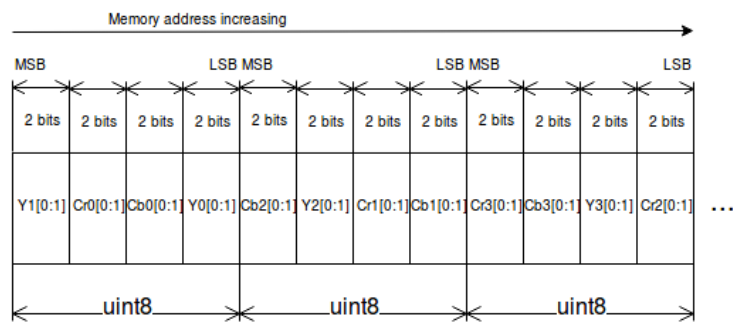
The image shall be packed row by row. The 8 MSB bits of all the color components of a row shall be stored in 1 block first, then the 2 LSB bits of all the color components of the row shall follow in a second block.



The 8 MSB bits shall be stored in the order Y-Cb-Cr:



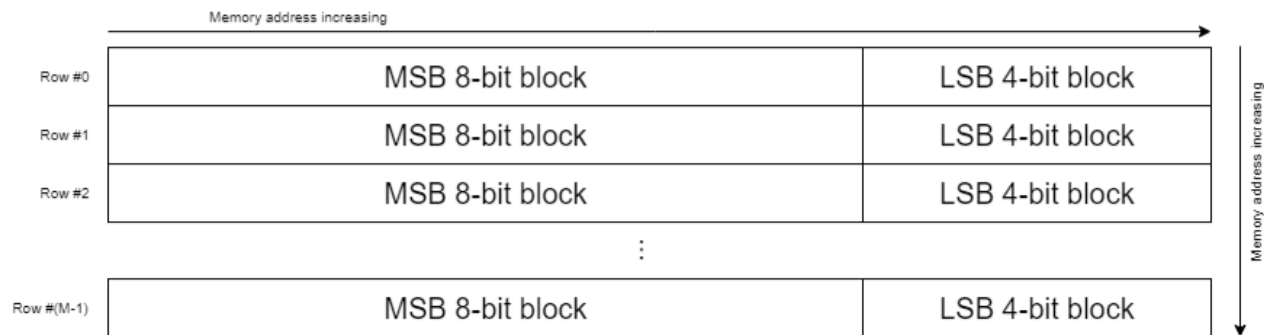
The 2 LSB bits shall also be stored in the order Y-Cb-Cr:



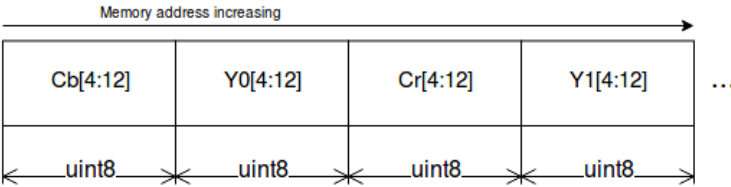
5.4.5 'y212'

Signal: YC_BCR, 4:2:2, 12 bit, interleaved

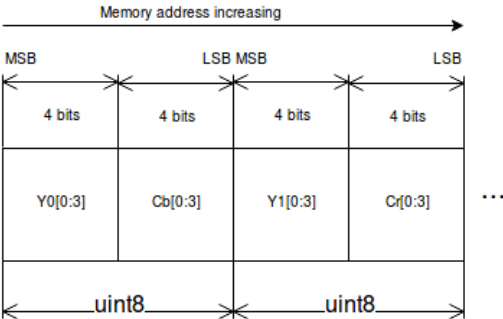
The image shall be packed row by row. Image width must be a multiple of 2. The 8 MSB bits of all the color components of a row shall be stored in 1 block first, then the 4 LSB bits of all the color components of the row shall follow in a second block.



The 8 MSB bits shall be stored in the order Cb-Y0-Cr-Y1:



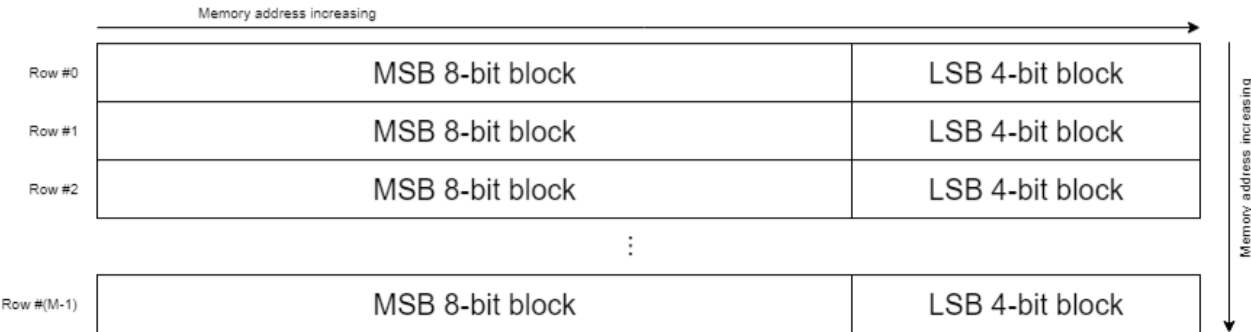
The 4 LSB bits shall also be stored in the order Cb-Y0-Cr-Y1:



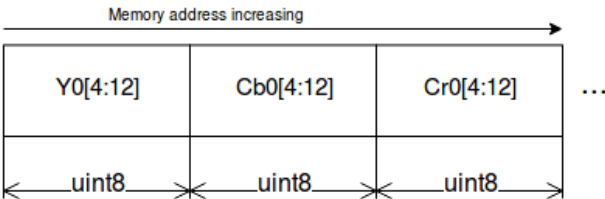
5.4.6 'y412'

Signal: YCbCr, 4:4:4, 12 bit, interleaved

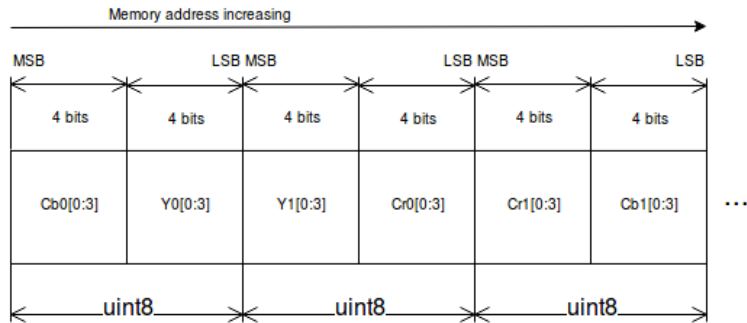
The image shall be packed row by row. The 8 MSB bits of all the color components of a row shall be stored in 1 block first, then the 4 LSB bits of all the color components of the row shall follow in a second block.



The 8 MSB bits shall be stored in the order Y-Cb-Cr:



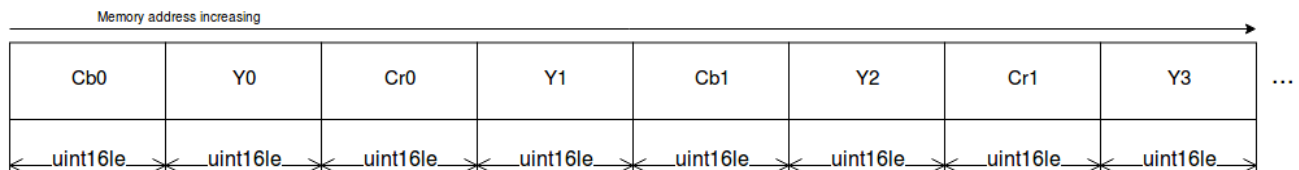
The 4 LSB bits shall be stored in the order Y-Cb-Cr at increasing 4-bit addresses:



5.4.7 'y216', 'y2S1', 'y2p6', 'y2p4'

Signal: YC_BCR, 4:2:2, 16 bit, S2.14 (16-bit), 10.6 (16-bit), 12.4 (16-bit), interleaved

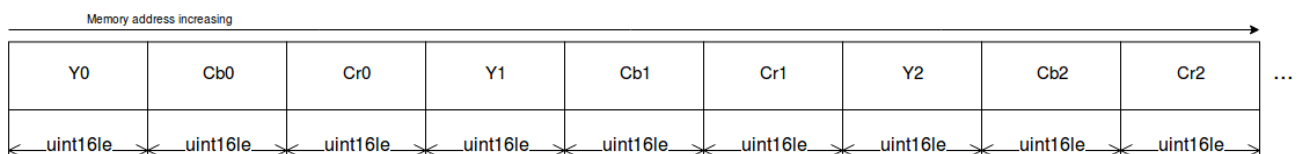
The image shall be packed row by row as 16-bit little-endian values. Image width must be a multiple of 2. The color components shall be packed in Cb0-Y0-Cr0-Y1 order. Each row of the image shall be padded to the nearest 8-byte boundary.



5.4.8 'y416', 'y4S1', 'y4p6', 'y4p4'

Signal: YC_BCR, 4:4:4, 16 bit, S2.14 (16-bit), 10.6 (16-bit), 12.4 (16-bit), interleaved

The image shall be packed row by row as 16-bit little-endian values. The color components shall be packed in Y-Cb-Cr order:



5.4.9 '\$20y2h'

Signal: YC_BCR, 4:2:2, short float (16 bit), interleaved

The image shall be packed row by row. Image width must be a multiple of 2. The color components shall be packed in Cb0-Y0-Cr0-Y1 order. Components shall be stored as IEEE 754-compliant 16-bit, little-endian float values.

5.4.10 '\$20y2f'

Signal: YC_BCR, 4:2:2, float (32 bit), interleaved

The image shall be packed row by row. Image width must be a multiple of 2. The color components shall be packed in Cb0-Y0-Cr0-Y1 order. Components shall be stored as IEEE 754-compliant 32-bit, little-endian float values.

5.4.11 ‘\$20y4h’

Signal: YCbCr, 4:4:4, short float (16 bit), interleaved

The image shall be packed row by row. The color components shall be packed in Y-Cb-Cr order. Components shall be stored as IEEE 754-compliant 16-bit, little endian float values.

5.4.12 ‘\$20y4f’

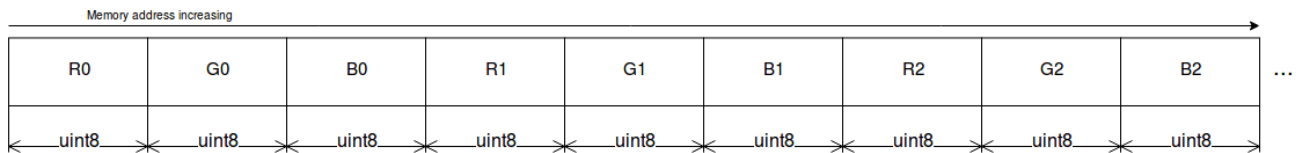
Signal: YCbCr, 4:4:4, float (32 bit), interleaved

The image shall be packed row by row. The color components shall be packed in Y-Cb-Cr order. Components shall be stored as IEEE 754-compliant 32-bit, little endian float values.

5.4.13 ‘rg08’

Signal: RGB, 8 bit, interleaved

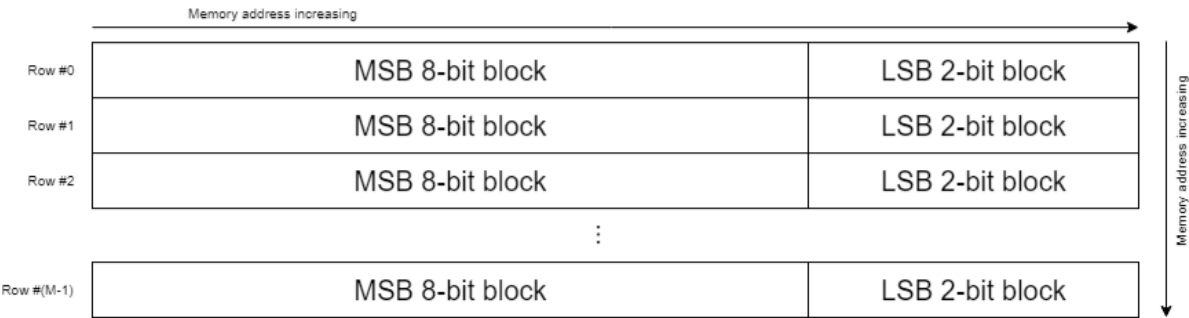
The image shall be packed row by row. The color components shall be packed in R-G-B order:



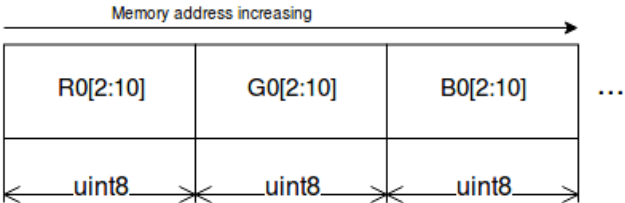
5.4.14 ‘rg10’

Signal: RGB, 10 bit, interleaved

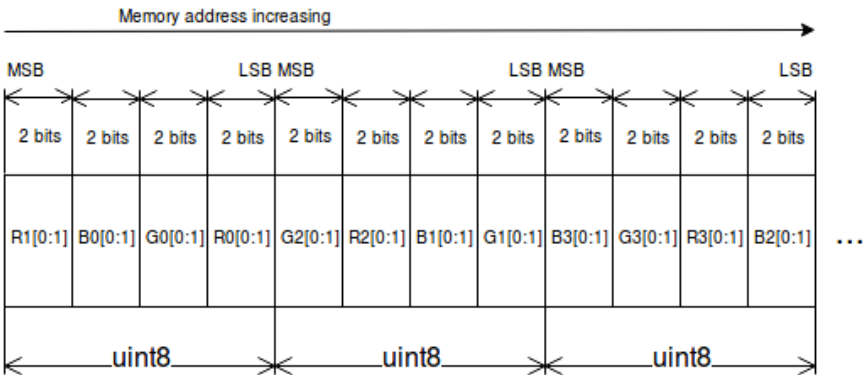
The image shall be packed row by row. The 8 MSB bits of all the color components of a row shall be stored in 1 block first, then the 2 LSB bits of all the color components of the row follow in a second block.



The 8 MSB bits are stored in R-G-B order



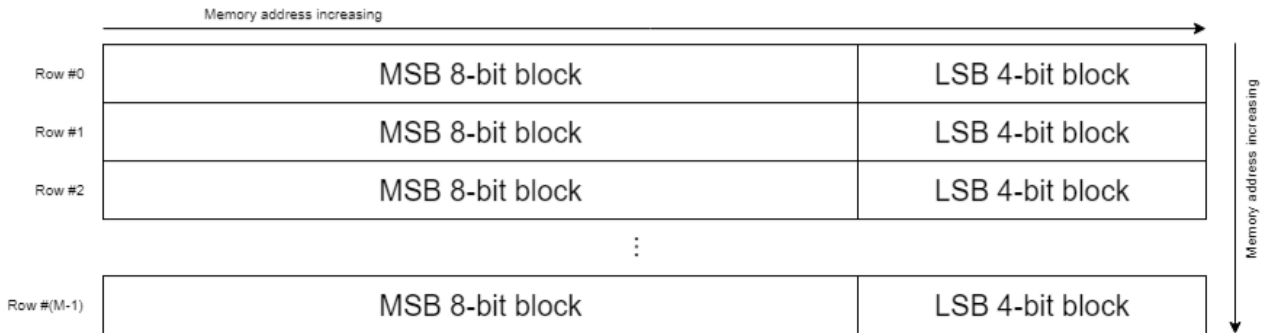
The 2 LBS bits shall be packed in R-G-B order and get padded at the end to a byte boundary



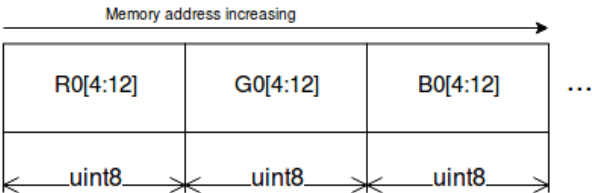
5.4.15 ‘rg12’

Signal: RGB, 12 bit, interleaved

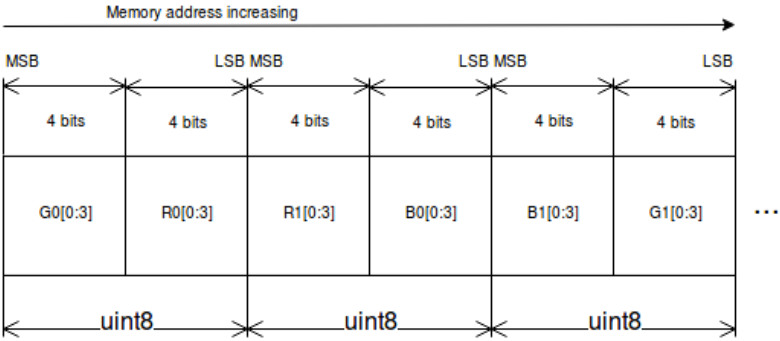
The image shall be packed row by row. The 8 MSB bits of all the color components of a row shall be stored in 1 block first, then the 4 LSB bits of all the color components of the row shall follow in a second block.



The 8 MSB bits shall be stored in the order R-G-B:



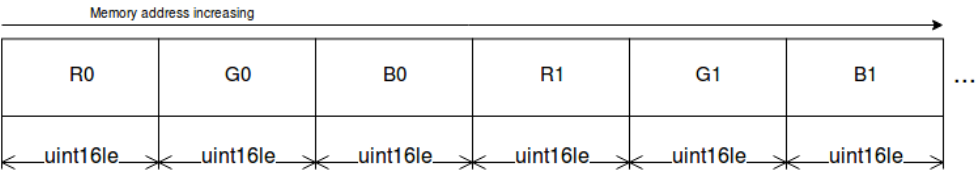
The 4 LSB bits shall be packed in R-G-B order at increasing 4-bit addresses with the block being padded at the end to a byte boundary:



5.4.16 ‘rg16’, ‘rgS1’, ‘rgp6’, ‘rgp4’

Signal: RGB, 16 bit, S2.14 (16-bit), 10.6 (16-bit), 12.4 (16-bit), interleaved

The image shall be packed row by row as 16-bit little-endian values. The color components shall be stored in R-G-B order:



5.4.17 ‘\$20rgh’

Signal: RGB, short float (16 bit), interleaved

The image shall be packed row by row. The color components shall be packed in R-G-B order. Components shall be stored as IEEE 754-compliant 16-bit, little endian float values.

5.4.18 ‘\$20rgf’

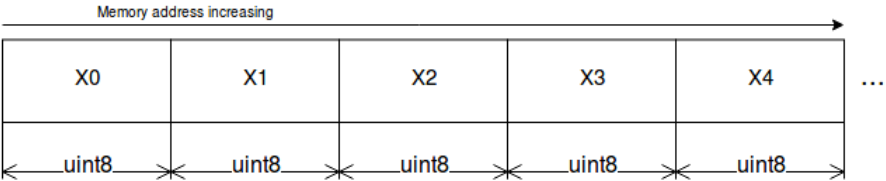
Signal: RGB, float (32 bit), interleaved

The image shall be packed row by row. The color components shall be packed in R-G-B order. Components shall be stored as IEEE 754-compliant 32-bit, little endian float values.

5.4.19 ‘\$20a08’, ‘\$20y08’, ‘cb08’, ‘cr08’, ‘\$20r08’, ‘\$20g08’, ‘\$20b08’

Signal: alpha, Y, C_B, C_R, R, G, B. 8 bit (plane)

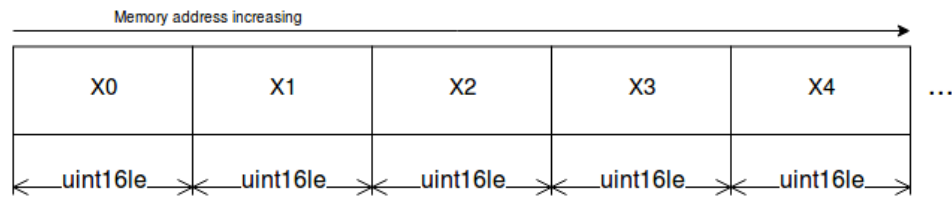
Each **fourcc** denotes a single-channel planar 8-bit representation of the image data. The image channel shall be packed row by row.



5.4.20 ‘\$20a16’, ‘\$20y16’, ‘cb16’, ‘cr16’, ‘\$20r16’, ‘\$20g16’, ‘\$20b16’

Signal: alpha, Y, C_B, C_R, R, G, B. 16 bit (plane)

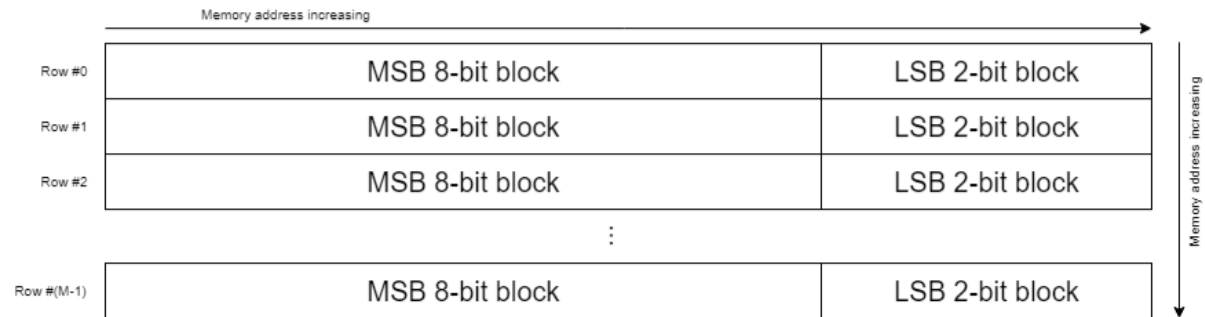
Each **fourcc** denotes a single-channel planar 16-bit representation of the image data. Each value shall be stored in little-endian form. The image channel shall be packed row by row.



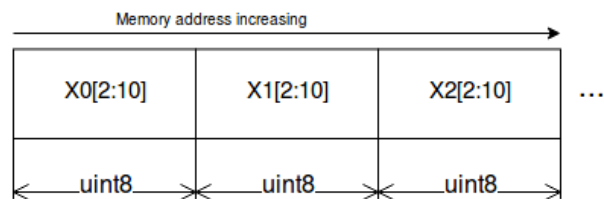
5.4.21 ‘\$20y10’, ‘cb10’, ‘cr10’, ‘\$20r10’, ‘\$20g10’, ‘\$20b10’

Signal: Y, C_B, C_R, R, G, B. 10 bit (plane)

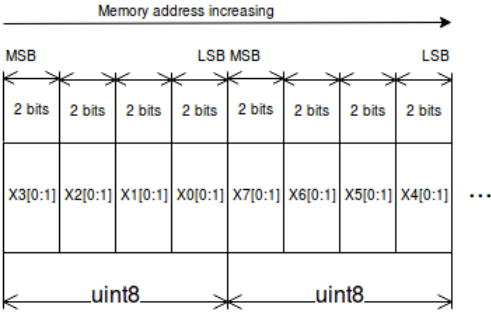
Each **fourcc** denotes a single-channel planar 10-bit representation of the image data. The image channel shall be packed row by row. The 8 MSB bits of a channel row shall be stored in 1 block first, then the 2 LSB bits of the channel row shall follow in a second block.



The 8 MSB bits of a channel row shall be stored sequentially one by one:



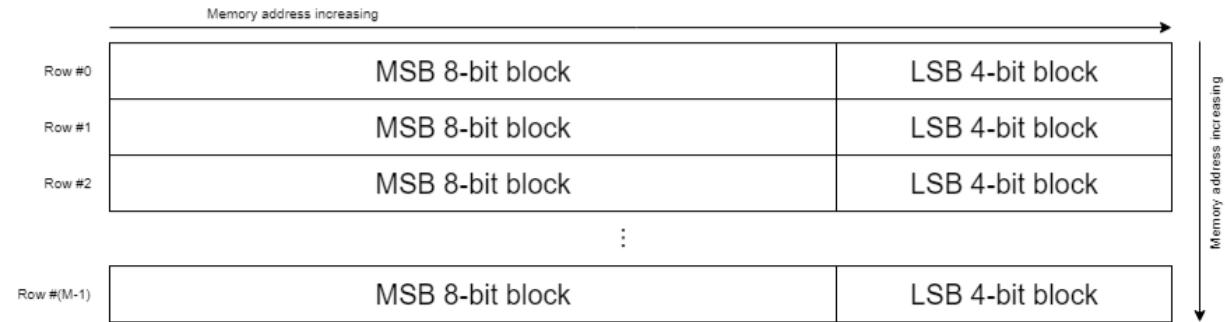
The 2 LSB bits of the channel row values shall be stored sequentially one by one and shall get padded at the end to a byte boundary



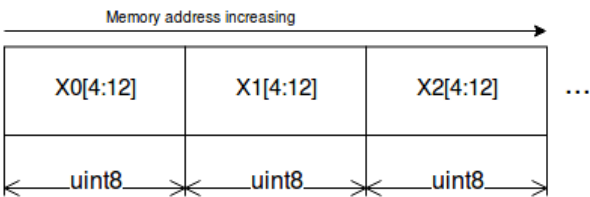
5.4.22 ‘\$20y12’, ‘cb12’, ‘cr12’, ‘\$20r12’, ‘\$20g12’, ‘\$20b12’

Signal: Y, C_B, C_R, R, G, B. 12 bit (plane)

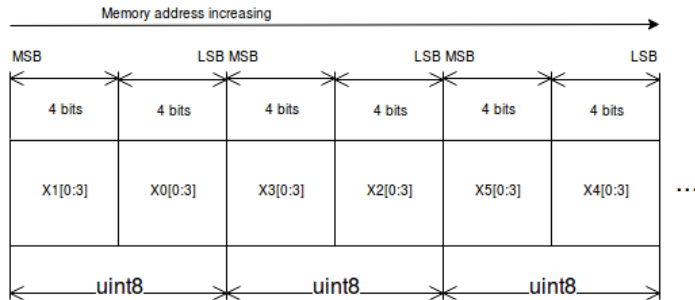
Each **fourcc** denotes a single-channel planar 12-bit representation of the image data. The image channel shall be packed row by row. The 8 MSB bits of a channel row shall be stored in 1 block first, then the 4 LSB bits of the channel row shall follow in a second block.



The 8 MSB bits of a channel row shall be stored sequentially one by one:



The 4 LSB bits of the channel row values shall be stored sequentially one by one and shall get padded at the end to a byte boundary



5.4.23 ‘\$20\$20yh’, ‘\$20cbh’, ‘\$20crh’, ‘\$20\$20rh’, ‘\$20\$20gh’, ‘\$20\$20bh’

Signal: alpha, Y, C_B, C_R, R, G, B. float16 (plane)

Each **fourcc** denotes a single-channel planar float16 representation of the image data. The image channel shall be packed row by row.

5.4.24 ‘\$20\$20yf’, ‘\$20cbf’, ‘\$20crf’, ‘\$20\$20rf’, ‘\$20\$20gf’, ‘\$20\$20bf’

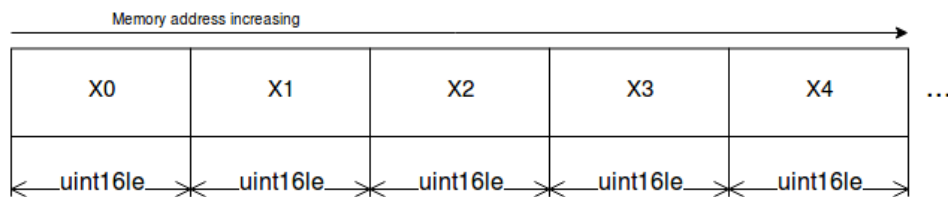
Signal: alpha, Y, C_B, C_R, R, G, B. float32 (plane)

Each **fourcc** denotes a single-channel planar float32 representation of the image data. The image channel shall be packed row by row.

5.4.25 ‘\$20yS1’, ‘\$20yp6’, ‘\$20yp4’, ‘cbS1’, ‘cbp4’, ‘cbp6’, ‘crS1’, ‘crp6’, ‘crp4’, ‘\$20r16’, ‘\$20rS1’, ‘\$20rp6’, ‘\$20rp4’, ‘\$20g16’, ‘\$20gS1’, ‘\$20gp6’, ‘\$20gp4’, ‘\$20b16’, ‘\$20bS1’, ‘\$20bp6’, ‘\$20bp4’

Signal: alpha, Y, C_B, C_R, R, G, B. S2.14 (16-bit), 10.6 (16-bit), 12.4 (16-bit) (plane)

Each **fourcc** denotes a single-channel planar 16-bit representation of the image data. The image channel shall be packed row by row as 16-bit little-endian values.



5.4.26 ‘rl08’, ‘rl16’

Signal: Alpha, 8 or 16 bit, RLE encoded (plane)

The RLE encoding scheme follows the algorithm described in SMPTE ST 2019-1, Section 7.3.1.2 with the following modification:

No macroblocks shall be used. Images shall be processed in rows, left to right, top to bottom. The run length shall only be limited by the number of elements in the slice.

The general structure of the ‘**sdat**’ payload is as shown in Figure 2

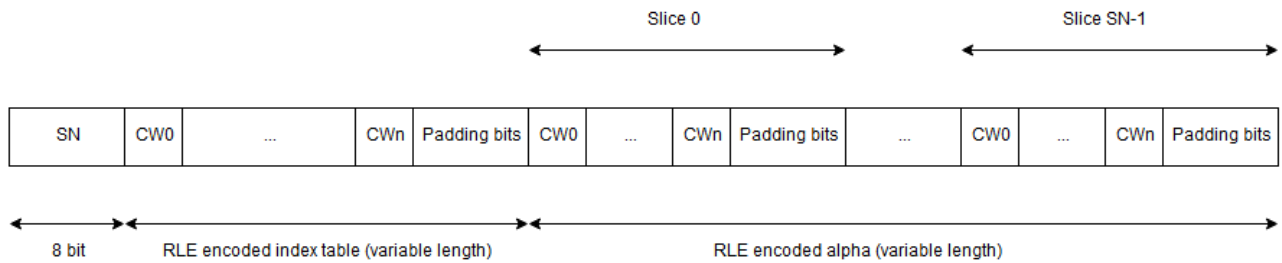


Figure 2 - RLE encoded alpha

Slice number SN

The alpha plane shall be broken down into slices to allow parallel processing. A slice shall be defined by its height (full rows). Slices shall be processed left to right, top to bottom. The number of slices shall be in the range [1..255] and is a choice of the encoder. The suggest default is 16.

Index table

The index table shall contain the size of the encoded slices, followed by their decoded heights.

Size 0	Size SN-1	Height 0	Height SN-1
--------	------	-----------	----------	------	-------------

Size 0 through SN-1 record the size, in bytes, of the compressed slice 0 through SN-1 respectively, while Height 0 through SN-1 records their height (in pixels). Slices may have different heights.

The index table shall be RLE encoded as a sequence of codewords according to the RLE scheme defined in SMPTE ST 2019-1, Section 7.3.1.2, Figure 35, with the following simplifications:

- No P bit is used and stored (implicitly always = 0)
- The bit depth D_{comp} of the elements is fixed to 24 bits

The index table shall be padded to the next byte boundary.

Slice coding

The alpha plane shall be encoded as a sequence of slices, top to bottom.

Every slice shall be encoded as a sequence of codewords. Every codeword shall be encoded according to SMPTE ST 2019-1, 7.3.1.2 with

- $D_{\text{comp}} = 8$ for 8 bit alpha ('r108')
- $D_{\text{comp}} = 16$ for 16 bit alpha ('r116')

If the compressed bitstream of the slice does not end on a byte boundary it shall be padded with zero bits to the next byte boundary.

Part 2 – Mapping DNxUncompressed to the MXF Generic Container

6 Essence wrapping of DNxPacked

6.1 Frame Wrapping (Informative)

The "Frame Wrapping" methods for DNxPacked (DNxUncompressed) byte-streams are shown in Figure 3 and Figure 4.

Figure 3 shows a series of images each wrapped in a single Content Package with no other Generic Container Elements in the Container. Each Content Package has the duration of one DNxPacked frame.

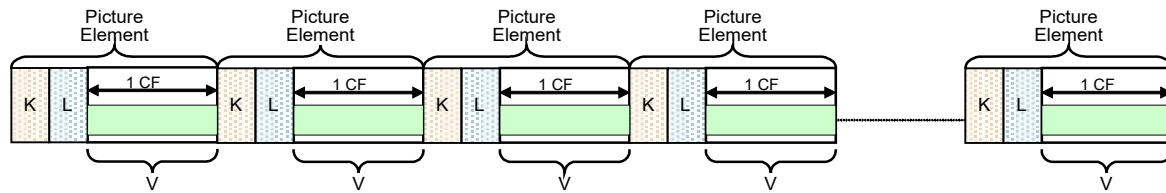


Figure 3 - Simple Representation of Frame Wrapping

The Frame Wrapping method enables Frame by Frame access by MXF applications which process at the KLV level. This can be particularly useful for applications which support multiple Generic Container mapping types, for example, MXF Operational Pattern OP-1A.

Sufficient Information is provided to allow individual Frames to be identified at the KLV level without an MXF decoder having to parse or decode the Essence Data. Each DNxPacked frame is KLV wrapped using a GC Picture Element Key.

In some applications, the Frame Wrapped DNxPacked picture data will exist in Content Packages with other Elements such as Sound and Data Elements. An example Generic Container is shown in Figure 4 below.

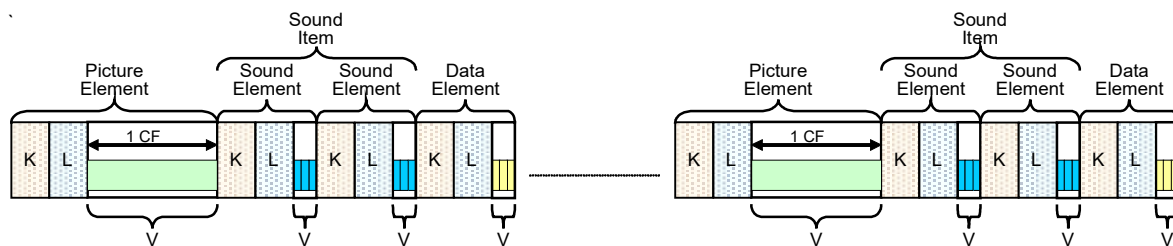


Figure 4 - Frame Wrapping with other CG Elements

The Generic Container Mapping Specifications for the Sound and Data Elements will detail the Key values and format of the data within the elements. In this wrapping mode, the Sound and Data Elements will follow the guidance given in SMPTE ST 379-2, Section 8.4 concerning placement of the synchronized samples in Essence Containers.

6.2 Clip Wrapping (Informative)

The "Clip Wrapping" methods for DNxPacked are shown in Figure 5 and Figure 6.

In Clip wrapping, KLV encoding wraps the whole of the DNxPacked stream that could contain a single frame or thousands of frames. Any other elements in the Generic Container could also be clip wrapped.

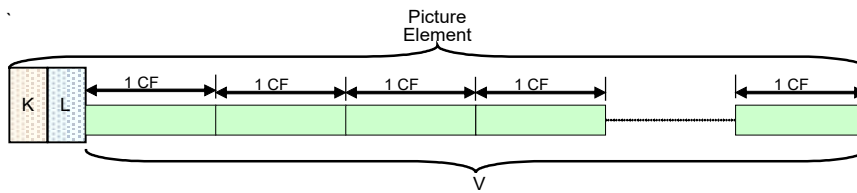


Figure 5 - Simple Representation of Clip Wrapping a Video Stream

The Clip Wrapping method is intended for applications which carry the DNxPacked stream as a single large entity. This can be very useful in applications where it is desired to use the rich metadata structures of MXF as an annotation to DNxPacked data and also in applications such as store and forward servers which process whole files.

The clip of DNxPacked data is KLV wrapped using an appropriate key as detailed in Section 6. When DNxPacked data is Clip wrapped, there could be only one clip per Generic Container. Single Clips can be wrapped as atomic MXF files, for example, using MXF Operational Pattern OP-Atom.

Multiple Clips can be concatenated and edited using the Operational Pattern mechanism detailed in the MXF format document. In some applications, the Clip Wrapped DNxPacked data will exist in a Content Package with other Elements such as Sound and Data Elements. An example Generic Container is shown in Figure 6.

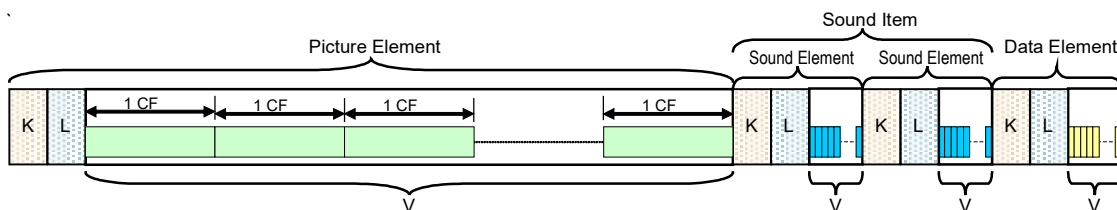


Figure 6 - Clip Wrapping with other GC Elements

The Generic Container Mapping Specifications for the Sound and Data Elements will detail the Key values and format of the data within the elements. Note that in this wrapping mode, the duration of Sound and Data Elements is intended to be that of the entire clip.

7 KLV Coding of DNxPacked Coding Units

7.1 Essence Element Key

The values of the bytes of the Essence Element Key are defined in SMPTE ST 379-2 Section 10.1 (Essence Element Key). The Picture Element Key shall be as specified in Table 3.

Byte #	Description	Value (hex)	Meaning
1~12	Defined in SMPTE ST 379-2	060E2B34.01020101.0D010301	See SMPTE ST 379-2
13	Item Type Identifier	15	GC Picture Item (as defined in SMPTE ST 379-2)
14	Essence Element Count	kk	Count of Picture Elements in the Picture item
15	Essence Element Type	18 19	Frame-wrapped DNxPacked Picture Element Clip-wrapped DNxPacked Picture Element
16	Essence Element Number	nn	The Number (used as an Index) of this Picture Element in the Picture Item
Label Description		Kind	URN
DNxPacked Picture Element Frame-wrapped		LEAF	urn:smpte:ul: 060E2B34.01020101.0D010301.15kk18nn
DNxPacked Picture Element Clip-wrapped		LEAF	urn:smpte:ul: 060E2B34.01020101.0D010301.15kk19nn

Table 3 - Key Values for the DNxPacked Picture Element Key

7.1.1 Essence Element Count – Byte 14

This shall be a count of the number of Picture Elements in the Picture Item of the Generic Container, typically 1.

7.1.2 Essence Element Type – Byte 15

The value of 0Ch identifies that each DNxPacked frame is frame-wrapped.

The value of 0Dh identifies that the sequence of DNxPacked frames are clip-wrapped.

7.1.3 Essence Element Number – Byte 16

This is a number used as an index to identify this instance of the Element Type within the Picture Item. Each Element within an Item shall have a unique value between 00h and 7Fh, as defined by SMPTE ST 379-2, which shall remain constant within the Generic Container.

7.2 Length

The length field should be 4 byte BER long-form encoded (i.e., 83h.xx.yy.zz) for frame-based wrapping and should be 8 byte BER long-form encoded (i.e., 87h.aa.bb.cc.dd.ee.ff.gg) for clip-based wrapping. Decoders shall conform to SMPTE ST 377-1.

The length field shall comply with SMPTE ST 379-2, Section 8.6.1.

7.3 Value

7.3.1 Frame-wrapped

The value field shall comprise a single DNxPacked 'pack' frame as defined in this document.

7.3.2 Clip-wrapped

The value field shall comprise a sequence of one or more concatenated DNxPacked ‘**pack**’ frames as defined in this document.

8 SMPTE Label for DNxPacked Essence Container Identification

The values for the Essence Container UL shall be as given in Table 4.

Byte #	Description	Value (hex)	Meaning
1-12	Defined by Generic Container	060E2B34.0401010D.0D010301	As defined in SMPTE ST 379-2
13	Essence Container Kind	02	MXF Generic Container
14	Mapping Kind	1E	DNxPacked Picture Element (as listed in SMPTE Labels Register)
15	Content Kind	01 02	MXF-GC Essence Container DNxPacked Frame Wrapped MXF-GC Essence Container DNxPacked Clip Wrapped
16	Reserved	00	
Label Description		Kind	URN
MXF-GC Essence Container DNxPacked		NODE	urn:smppte:ul: 060E2B34.0401010D.0D010301.021E0000
MXF-GC Essence Container DNxPacked Frame Wrapped		LEAF	urn:smppte:ul: 060E2B34.0401010D.0D010301.021E0100
MXF-GC Essence Container DNxPacked Clip Wrapped		LEAF	urn:smppte:ul: 060E2B34.0401010D.0D010301.021E0200

Table 4 – Specification of the DNxPacked Essence Container Labels

The Essence Container UL is used within a batch of ULs in Partition Packs and the Preface set and on its own in the Essence Descriptor. These UL values are listed in the SMPTE Labels Register.

9 SMPTE Label for DNxUncompressed Picture Essence Coding

The values for the Picture Essence Coding UL shall be as given in Table 5.

Byte #	Description	Value (hex)	Meaning
1-8	Registry Designator	060E2B34.0401010D	Designator value is defined in SMPTE ST 400
9	Parametric	04	Node used to define parametric data
10	Picture Essence	01	Identifies picture essence coding
11	Picture Coding Characteristics	02	Identifies picture coding characteristics
12	Compressed Picture Coding	02	Identifies compressed picture coding
13	Individual Picture Coding Schemes	03	Identifies individual picture coding schemes
14	DNxUncompressed Picture Coding	07	DNxUncompressed Picture Coding
15	DNxUncompressed Standard Formats DNxUncompressed S2.14, 10.6, 12.4 formats	01 02	DNxUncompressed Picture Coding: Standard formats DNxUncompressed Picture Coding: S2.14, 10.6, 12.4 formats
16	unused	00	Unused
Label Description		Kind	URN
DNxUncompressed Picture Coding		NODE	urn:smpte:ul: 060E2B34.0401010D.04010202.03070000
DNxUncompressed Picture Coding: Standard formats		LEAF	urn:smpte:ul: 060E2B34.0401010D.04010202.03070100
DNxUncompressed Picture Coding: Fixed point formats		LEAF	urn:smpte:ul: 060E2B34.0401010D.04010202.03070200

Table 5 – Specification of the DNxUncompressed Picture Essence Coding Label

The Picture Essence Compression UL is used in the Generic Picture Essence Descriptor. This UL is listed in the SMPTE Labels Register.

Byte 15 shall be set to 01h for all formats not involving S2.14, 10.6 or 12.4 components.

Byte 15 shall be set to 02h if the image components (except alpha) use the S2.14, 10.6 or 12.4 format. All image components (except alpha) shall be of the same fixed-point format. See CDCI Picture Descriptor and RGBA Picture Descriptor for more details.

A Picture Coding Variant code of 02h shall not be used with any standard float 10- or 12-bit component values. All other fourcc codes are permitted in either variant.

10 Essence Descriptors for DNxUncompressed

The File Descriptor sets are those structural metadata sets in the Header Metadata that describe the essence and metadata elements defined in this document. The structure of these sets is defined in the MXF File Format Specification (SMPTE ST 377-1) and in some Generic Container mapping specifications. File Descriptor sets shall be present in the Header Metadata for each Essence Element.

The mapping of DNxUncompressed uses the CDCI (Color Difference Component Image) Picture Essence Descriptor or RGBA (Red Green Blue Alpha) Picture Essence Descriptor as defined in SMPTE ST 377-1, Annex F.4. The meaning of the items in the CDCI Picture Essence Descriptor and RGBA Picture Essence Descriptor (and their super-classes) are defined in SMPTE ST 377-1, Annex G.

10.1 Generic Picture Essence Descriptor

Item Name	Meaning	Source of derived data
Frame Layout	Interlaced or Progressive layout	<code>signal_info_box.frame_layout</code>
Stored Width	Horizontal Size of stored picture	<code>signal_info_box.frame_width</code>
Stored Height	Vertical Size of stored picture	<code>signal_info_box.frame_height</code>

Table 6 - Generic Picture Essence Descriptor Values

The Frame Layout shall be constrained to the values FULL_FRAME (0) for progressive and MIXED_FIELDS (3) for interlaced frames.

Informative note: The Stored Height is not necessarily identical to `signal_info_box.frame_height`, but needs to take the Frame Layout into account (see ST 377-1:2011, G.2.7)

10.2 CDCI Picture Essence Descriptor

Item Name	Meaning	Source of derived data	Legal values
Component Depth	Number of active bits per sample	<code>signal_info_box.packed_type</code>	8, 10, 12, 16, 253 (half-float), 254 (float)
Horizontal Subsampling	Specifies the H color subsampling		1,2
Vertical Subsampling	Specifies the V color subsampling		1,2
Alpha Sample Depth	Number of bits per alpha sample		8,16

Table 7 – CDCI Picture Essence Descriptor Values

For planar representations of CDCI Picture Essence all non-Alpha components shall use the same Component Depth.

The S2.14 format is treated as a float format and the Component Depth shall be signaled as 254 (Picture Coding Variant 02h only). A client can decode S2.14 as either float or S2.14 values.

The 10.6 format is treated as a 10-bit format and the Component Depth shall be signaled as 10 (Picture Coding Variant 02h only). A client can decode 10.6 as either 10-bit (dropping the 6 LSB bits) or 10.6 values.

The 12.4 format is treated as a 12-bit format and the Component Depth shall be signaled as 12 (Picture Coding Variant 02h only). A client can decode 12.4 as either 12-bit (dropping the 4 LSB bits) or 12.4 values.

10.3 RGBA Picture Essence Descriptor

Item Name	Meaning	Source of derived data	Comment
PixelLayout	Pixel quantization and order as a data structure	signal_info_box.packed_type of all packed ' sinf ' image boxes	The component order shall match the enumeration order in ' pack '

Table 8 - RGBA Picture Essence Descriptor Values

The S2.14 format is treated as a float format and the Depth item shall be signaled as 254 (Picture Coding Variant 02h only). A client can decode S2.14 as either float or S2.14 values.

The 10.6 format is treated as a 10-bit format and the Depth item shall be signaled as 10 (Picture Coding Variant 02h only). A client can decode 10.6 as either 10-bit (dropping the 6 LSB bits) or 10.6 values.

The 12.4 format is treated as a 12-bit format and the Depth item shall be signaled as 12 (Picture Coding Variant 02h only). A client can decode 12.4 as either 12-bit (dropping the 4 LSB bits) or 12.4 values.

All other items shall follow SMPTE ST 377-1, Annex F (Specification of Descriptors used in MXF) and Annex G (Picture Essence Descriptor Properties).

11 Application Issues - Alignment

As noted in SMPTE ST 377-1, Section 5.4.1, "it may be desirable to align certain KLV elements to specific byte boundaries. This can be achieved with the insertion of KLV Fill items to ensure the desired items are aligned."

For DNxUncompressed MXF encoders should place the first byte of a '**pack**' element on a 256 byte boundary in the file by insertion of KLV fill items in accordance with SMPTE ST 377-1, Section 5.4, as shown in Figure 6 of SMPTE ST 377-1. Any further alignment requirements inside the '**pack**' box then need to be handled by the essence encoder via the use of '**fill**' boxes within the '**pack**' element.

This alignment should be signaled by placing the value 256 in the Image Alignment Offset property of the Picture Essence Descriptor.

For Frame Wrapping, the alignment should be achieved by inserting an initial KLV Fill before the first Frame-Wrapped GC Content Package, and ending each Frame-Wrapped GC Content Package with a KLV Fill.

For Clip-Wrapping, only an initial KLV Fill is required. Later '**pack**' elements need to be aligned within the byte-stream by adding appropriate '**fill**' elements at the end of the '**pack**' elements. To avoid any requirement to insert unneeded KLV Fill items throughout the file, the mapping of DNxUncompressed need not use KAG in the partition pack.

12 Bibliography

ISO/IEC 14496-12:2015: Information Technology - Coding of audio-visual objects – Part 12: ISO base media file format