

Pixel Format Naming Convention

(PFNC)

Version 2.5

Table of Content

1 Introduction 7
1.1 Purpose 7
1.2 Definitions and Acronyms 8
1.2.1 Definitions 8
1.2.2 Acronyms 8
1.3 Reference Documents 10
1.4 Assumptions 10
2 Summary of the Pixel Naming Convention 12
3 Components and Location 14
3.1 Pixel Location in Image 14
3.1.1 Mono Location 14
3.1.2 LMN444 Location 14
3.1.3 LMN422 Location 15
3.1.4 LMN411 Location 16
3.1.5 LMNO4444 Location 16
3.1.6 LM44 Location 17
3.1.7 Bayer Location 17
3.1.8 BiColor_LMNO Location 19
3.1.9 Sparse Color Filter Location 20
3.1.10 CFA_xxxx Location (square pattern) 22
3.1.11 CFA<#lines>by<#columns>_xxxx Location (non-square pattern) 23
3.1.12 Polarized Location 24
3.2 Components 25
3.2.1 CFA Basic Components 29
3.3 Generic Data Types Formats 29
4 Number of bits for each component 30
5 Optional “Data type” indicator 31

6 Optional Packing Style 32
6.1 Unpacked 32
6.1.1 lsb Unpacked 32
6.1.2 msb Unpacked 33
6.2 Cluster marker 34
6.3 Packed tag 35
6.3.1 lsb Packed 35
6.3.2 msb Packed 36
6.4 Grouped tag 37
6.4.1 lsb Grouped 38
6.4.2 msb Grouped 39
6.5 Align tag 40
6.6 Packing Style Summary 42
6.7 Dealing with Line and Image Boundaries 43
7 Interface-specific 44
7.1 Planar mode 44
7.2 Semiplanar mode 44
7.3 Components Sequencing 45
8 Appendix A - Color Space Transforms 46
8.1 Gamma Correction 46
8.2 Y'CbCr Conversions 47
8.2.1 Generic Full Scale Y'CbCr (8-bit) 47
8.2.2 Y'CbCr601 (8-bit) 49
8.2.3 Y'CbCr709 (8-bit) 52
9 Appendix B - Sub-sampling notation 55
9.1 Co-sited Positioning 55
9.2 Centered Positioning 56
10 Appendix C - Pixel Format Value Reference 57
11 Document History 58

List of Figures

Figure 1-1 : 8-bit pixel data 10
Figure 1-2 : 16-bit pixel data 10
Figure 1-3 : 32-bit pixel data 10
Figure 2-1 : Naming Convention Text Fields 12
Figure 3-1: Mono Pixel Location 14
Figure 3-2: LMN444 Pixel Location 15
Figure 3-3: LMN422 Pixel Location 15
Figure 3-4: LMN411 Pixel Location 16
Figure 3-5: LMNO4444 Pixel Location 16
Figure 3-6 : LM44 Pixel Location 17
Figure 3-7: BayerRG array 17
Figure 3-8: Bayer_LMMN Pixel Location 17
Figure 3-9: BayerBG array 18
Figure 3-10: Bayer_NMML Pixel Location 18
Figure 3-11: BayerGR array 18
Figure 3-12: Bayer_MLNM Pixel Location 19
Figure 3-13: BayerGB array 19
Figure 3-14: Bayer_MNLM Pixel Location 19
Figure 3-15: Bi-color sensor with 2 stages 20
Figure 3-16: Bi-color camera output from 2 stages 20
Figure 3-17: BiColor_LMNO Pixel Location 20
Figure 3-18: SCF1_LMLN Pixel Location 21
Figure 3-19: SCF1_LNLM Pixel Location 21
Figure 3-20: SCF1_LOLN Pixel Location 22
Figure 3-21: SCF1_LNLO Pixel Location 22
Figure 3-22 : Examples of a generic 4x4 CFA 23
Figure 3-23: CFA1by4_GRGB array 23
Figure 3-24 POL22M1 array 24
Figure 3-25: POL44C1 array 25

Figure 6-1: Mono8 unpacked 32
Figure 6-2:Mono10 unpacked 32
Figure 6-3: Mono12 unpacked 33
Figure 6-4:Mono10msb unpacked 33
Figure 6-5: Mono12msb unpacked 33
Figure 6-6 : 10-bit monochrome pixel lsb packed into 32 bits (Mono10c3p32) 34
Figure 6-7 :3 components in 10-bit lsb packed into 32-bit pixel (RGB10p32) 35
Figure 6-8 :3 components lsb packed into 16-bit pixel (RGB565p) 35
Figure 6-9 : 10-bit monochrome pixel lsb packed (Mono10p) 36
Figure 6-10 :3 components in 10-bit msb packed into 32-bit pixel (RGB10p32msb) 37
Figure 6-11 : 10-bit monochrome pixel msb packed (Mono10pmsb) 37
Figure 6-12: 2 monochrome 10-bit pixels with lsb grouped into 12 bits (Mono10g12) 38
Figure 6-13: 2 monochrome 12-bit pixels with lsb grouped into 24 bits (Mono12g) 38
Figure 6-14: 3 components of 10-bit with lsb grouped into 32-bit pixel (RGB10g32) 38
Figure 6-15 : 3 components of 12-bit with lsb grouped into 40-bit pixel (RGB12g40) 38
Figure 6-16 : 3 components of 10-bit with msb grouped into 32-bit pixel (RGB10g32msb) 39
Figure 6-17: RGB 8-bit unpacked aligned to 32-bit (RGB8a32) 40
Figure 6-18 : Using a cluster marker of 3 unpacked Mono10 aligned to 64 bits (Mono10c3a64) 40
Figure 7-1: RGB10_Planar 44
Figure 8-1: Gamma Correction for ITU-R BT.601 (image from Wikipedia) 46
Figure 8-2 : Generic full scale Y'CbCr 48
Figure 8-3 : Full scale RGB for BT.601 50
Figure 8-4 : Scaled down rgb for BT.601 51
Figure 8-5 : Full scale RGB for BT.709 53
Figure 8-6 : Scaled down rgb for BT.709 54
Figure 9-1 : Chroma positioning (co-sited alignment) 55
Figure 9-2 : Chroma positioning (centered alignment) 56

List of Equations

Equation 1 : Gamma Correction 46
Equation 2 : Generic full scale R'G'B' to Y'CbCr conversion (8 bits) 49
Equation 3 : Generic full scale Y'CbCr to R'G'B' conversion (8 bits) 49
Equation 4 : Full scale R'G'B' to Y'CbCr601 conversion (8 bits) 50
Equation 5 : Full scale Y'CbCr601 to R'G'B' conversion (8 bits) 51
Equation 6 : Scaled down r'g'b' to Y'CbCr601 conversion (8 bits) 51
Equation 7 : Y'CbCr601 to r'g'b' conversion (8 bits) 52
Equation 8 : Full scale R'G'B' to Y'CbCr709 conversion (8 bits) 53
Equation 9 : Full scale Y'CbCr601 to R'G'B' conversion (8 bits) 53
Equation 10 : Scaled down r'g'b' to Y'CbCr709 conversion (8 bits) 54
Equation 11 : Y'CbCr709 to R'G'B' conversion (8 bits) 54

1 Introduction

1.1 Purpose

The intention of this document is to define a generic convention to name the pixel formats used in machine vision. This covers 2D images as well as 3D imaging data. The aim is not to provide a unique definition for all theoretical possibilities, but to provide clear guidelines to follow when a new pixel format is introduced. As such, the pixel format designation is not sufficient to deduce all the pixel characteristics (that would be next to impossible anyway with the number of possible permutations!), but following those guidelines should provide a uniform way to name new pixel types so they fit well within the current set, even though the layout of each specific pixel format might need to be explicitly illustrated. When this convention is not sufficient, a camera interface-specific designator can be appended to remove any ambiguity.

Note: The main objective is to have clear guidelines in how to designate pixel format: a text string associated to a pixel format. The actual numerical value associated to each pixel format, the GenICam display name and the way pixel information is put into data packets is beyond the scope of this document.

This document covers the traditional 2D images, but starting with version 2.0 it also introduces support for 3D imaging data. For 3D, the formats are proposed as “abstract” with no defined mapping to actual real-world units and coordinate systems (such as Cartesian or spherical) or its properties (such as orientation). Such mapping should be defined by other means, in particular through the GenICam Standard Feature Naming Convention (SFNC) device description file. Letters A, B and C are used for the abstract coordinate names, where A-B-C can mean X-Y-Z for the Cartesian coordinate system, Theta-Phi-Rho for the spherical system, etc. For so-called 2.5D, the C always stands for the “depth/range” coordinate that can also be transferred standalone.

The Pixel Format Naming Convention supplements the GenICam Standard Feature Naming Convention (SFNC). As such, it is a child document of the SFNC. Request for clarifications or to add pixel formats not supported by the current syntax should be directed at the current editor of this document, as listed in the document history section.

Important: The PFNC defines pixels names and formats in an interoperable way so they can be shared across technologies. A given imaging standard might use internally a different low level bit encoding while keeping the PFNC name. This is acceptable as long as the low level bit encoding respects PFNC when it exits the boundary of that imaging standard. In that case, the imaging standard is required to define its actual bit coding.

1.2 Definitions and Acronyms

1.2.1 Definitions

Cluster	A group of single-component/monochrome pixels combined together and treated as a multi-component pixel.
Component	One of the constituents necessary to uniquely represent a pixel value. For monochrome pixels, only one component is necessary (ex: luma). For color pixels, multiple components might be needed (ex: red-green-blue or Y'CbCr). For 3D data pixels, one or more components expressing the pixel coordinate would be used.
Pixel	A single point in an image that can contain more than one component.
Bayer	A specific type of color filter array using a 2x2 tile with 1 red, 2 green and 1 blue components.

1.2.2 Acronyms

a	Alpha component
A	First component for 3D data pixel
AIA	Imaging association based in the USA
b	Scaled down blue color component (ex: 235 values in 8-bit, must be specified by the standard referencing the Pixel Format Naming Convention)
B	Full scale blue color component (ex: 256 values in 8-bit) or second component for 3D data pixel
C	Third component for 3D data pixel
Cb	Chroma blue
CFA	Color Filter Array
Cr	Chroma red
Cw	Full scale custom wavelength component
EMVA	European Machine Vision Association
FourCC	Four-Character Code
g	Scaled down green color component (ex: 235 values in 8-bit, must be specified by the standard referencing the Pixel Format Naming Convention)
G	Full scale green color component (ex: 256 values in 8-bit)
HDTV	High Definition Television

IIDC	The 1394 Trade Association Instrumentation and Industrial Control Working Group, Digital Camera Sub Working Group
Ir	Infrared color component
IR	Infrared
ITU	International Telecommunication Union
JIIA	Japanese Industrial Imaging Association
JPEG	Joint Photographic Experts Group
L	Generic first component of a pixel
lsb	Least significant bit
LSB	Least Signficant Byte
M	Generic second component of a pixel
MPEG	Moving Picture Experts Group
msb	Most significant bit
MSB	Most Significant Byte
N	Generic third component of a pixel
O	Generic fourth component of a pixel
r	Scaled down red color component (ex: 235 values in 8-bit, must be specified by the standard referencing the Pixel Format Naming Convention)
R	Full scale red color component (ex: 256 values in 8-bit)
SDTV	Standard Definition Television
Y'	Luma
U	1st chroma in YUV (blue – luma color difference)
V	2nd chroma in YUV (red – luma color difference)
W	White color component (equivalent to monochrome)

1.3 Reference Documents

IEC 60559:1989	Binary floating-point arithmetic for microprocessor systems, second edition (IEC 60559:1989), also known as IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE 754−1985)
GenICam	Generic Interface for Camera, version 3.0
GenICam SFNC	GenICam Standard Features Naming Convention, version 2.2
ITU-R BT.601-7	Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios
ITU-R BT.709-5	Parameter values for the HDTV standards for production and international programme exchange
JFIF	JPEG File Interchange Format, version 1.02

1.4 Assumptions

- Pixels have a maximum of 4 components (ex: alpha-red-green-blue). In this text, we use the generic LMNO designation to represent those components (ex: LMN could represent RGB where R = L, G = M and B = N).
- Some components might be sub-sampled (ex: Y'CbCr 4:2:2 and 4:1:1).
- The following figure illustrates 8-bit, 16-bit and 32-bit data words respectively. The way this data is stored in memory (little-endian or big-endian) is not defined by this convention, though the illustrations use little-endian.

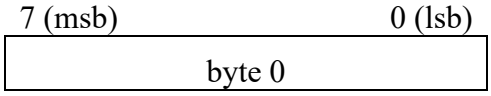


Figure 1-1 : 8-bit pixel data

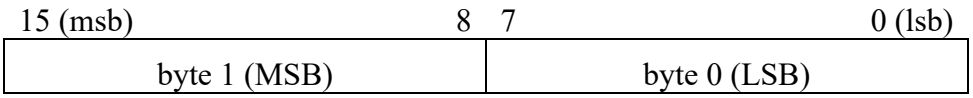


Figure 1-2 : 16-bit pixel data

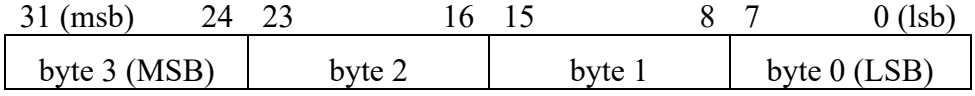


Figure 1-3 : 32-bit pixel data

Important: The standard referencing this Pixel Format Naming Convention is expected to define in their document if little-endian or big-endian is used (when necessary).

- A **cluster** represents a group of successive single-component/monochrome pixels put together and considered as one unit for alignment purpose. This can be used to align single-component/monochrome data to a given boundary, such as 32-bit or 64-bit, when at least a full byte of zeros is used for padding. This allows re-using some of the multi-components/color pixel packing concepts for a group of single-component/monochrome pixels. A cluster is considered a multi-component pixel.
- Historically, YUV is the standard color space used for analog television transmission, while Y'CbCr is used for digital encoding of color information suited for video and still-image compression and transmission such as JPEG and MPEG. However, the YUV nomenclature is now used rather loosely and many times incorrectly refers to digital components. This naming convention recognizes this mismatch. It nevertheless refers to YUV in some situations, even though Y'CbCr would be the appropriate representation, as this mismatch has widespread usage in the industry. Therefore, this text assumes that YUV is a general term for a color space working along the principles of Y'CbCr.

2 Summary of the Pixel Naming Convention

A pixel name is a text string composed of the following 5 fields, the last 3 having default values when they are not explicitly indicated.

Components & Location	# bits	[data type]	[packing]	[interface-specific]

Figure 2-1 : Naming Convention Text Fields

Table 2-1 : Naming Convention Text Fields

Field	Description
Components and Location	Provides the list of components (ex: RGB, Y'CbCr, abstract A-B-C for 3D, …) and a reference to pixel location/sub-sampling if needed (ex: BayerRG, Y'CbCr422, …). In certain cases, an identifier might be used to differentiate between 2 similar color formats (Y'CbCr using ITU-R BT.601 vs ITU-R BT.709).
# bits	# of bits of each component
Data type (optional)	Data type indicator • *empty* or '**u**': unsigned integer data • '**s**': 2's complement signed integer data • '**f**': IEC 60559:1989 compliant floating point data
Packing (optional)	Packing style indicator showing how data is put into bytes and how to align them. • *empty*: unpacked data. Empty bits of each component must be padded with 0 to align to byte boundary. • '**p**': packed data with no bit left in between components. • '**g**': grouped data where least significant bits or most significant bits of the components are grouped in a separate byte. • '**c**': cluster of single-component/monochrome pixels indicating the number of pixels to put together. This marker does not provide packing information per say. • '**a**': an additional tag indicating the pixel is aligned to the given number of bits.
Interface-specific (optional)	This field is specific to the camera interface. It is the responsibility of the specific standard to define how to use this field. For instance, this field could be used to specify how data is ordered into data packets (sequencing of components in the packet) or on various image streams (ex: planar mode).

By concatenating this information, it is possible to create a text string that can uniquely describe a pixel format.

Rule: If a pixel name requires 2 numbers in its designation as part of consecutive fields, then they must be separated by an underscore (‘_’). Otherwise, no underscore is used.

Ex: YCbCr709_422_8 for 8-bit per component Y’CbCr 4:2:2 using ITU-R BT.709.

The following chapters describe in details each of these fields.

3 Components and Location

The Components field provide the list of component constituents available in the pixel format. Location offers additional information regarding the positioning of the components in the image. The combination of the two gives a good idea of the pixel format as seen by the user.

3.1 Pixel Location in Image

This section lists the various components positioning within the image. It is especially helpful when sub-sampling of certain components is used. This information is required to determine the "Components and Location" field of the pixel name.

In the following diagrams, for a given pixel, the first index represents the row number; the second index represents the column number.

The figures of this section uses generic pixel component format where 'L' represents the first component, 'M' the second, 'N' the third and 'O' the fourth (if necessary). To help clarify some of them, you can think about LMN = RGB (where R = L, G = M and B = N) or LMN = Y'CbCr (where Y' = L, Cb = M and Cr = N). Same hold true for Bayer patterns (where R = L, G = M and B = N).

3.1.1 Mono Location

This format is used for single component images where typically L is the luma (Y'). This could also be used for planar transfer where each component of the pixel is separated onto a different stream.

Ex: Mono8

	1	2	3	4	…
1	L_{11}	L_{12}	L_{13}	L_{14}	…
2	L_{21}	L_{22}	L_{23}	L_{24}	…
3	L_{31}	L_{32}	L_{33}	L_{34}	…
4	L_{41}	L_{42}	L_{43}	L_{44}	…
…	…	…	…	…	…

Figure 3-1: Mono Pixel Location

3.1.2 LMN444 Location

This format is typically used for any 3 component color space, such as RGB and Y'CbCr. No sub-sampling is performed.

Ex: RGB8

	1	2	3	4	…
1	LMN_{11}	LMN_{12}	LMN_{13}	LMN_{14}	…
2	LMN_{21}	LMN_{22}	LMN_{23}	LMN_{24}	…
3	LMN_{31}	LMN_{32}	LMN_{33}	LMN_{34}	…
4	LMN_{41}	LMN_{42}	LMN_{43}	LMN_{44}	…
…	…	…	…	…	…

Figure 3-2: LMN444 Pixel Location

3.1.3 LMN422 Location

This format is a 4:2:2 co-sited sub-sampled representation of a 3 component color space. The M and N components are sub-sampled by 2 horizontally: their effective positions are co-sited with alternate L samples, starting in the first column.

Ex: YCbCr422_8

	1	2	3	4	…
1	LMN_{11}	L_{12}	LMN_{13}	L_{14}	…
2	LMN_{21}	L_{22}	LMN_{23}	L_{24}	…
3	LMN_{31}	L_{32}	LMN_{33}	L_{34}	…
4	LMN_{41}	L_{42}	LMN_{43}	L_{44}	…
…	…	…	…	…	…

Figure 3-3: LMN422 Pixel Location

When 4:2:2 sub-sampling is used, the components are transmitted using the following order, unless a component order is explicitly stated in the standard referencing the Pixel Format Naming Convention.

L_{11} , M_{11} , L_{12} , N_{11} , L_{13} , M_{13} , L_{14} , N_{13} …

The above component order is equivalent to FourCC[1] YUY2.

[1] FourCC is short for “four-character code”, an identifier for a video codec, compression format, color or pixel format used in media file.

3.1.4 LMN411 Location

This format is a 4:1:1 co-sited sub-sampled representation of a 3 component color space. The M and N components are sub-sampled by 4 horizontally and are thus associated to 4 consecutive columns. Their position is co-sited starting with the first L sample.

Ex: YCbCr411_8

	1	2	3	4	5	…
1	LMN_{11}	L_{12}	L_{13}	L_{14}	LMN_{15}	…
2	LMN_{21}	L_{22}	L_{23}	L_{24}	LMN_{25}	…
3	LMN_{31}	L_{32}	L_{33}	L_{34}	LMN_{35}	…
4	LMN_{41}	L_{42}	L_{43}	L_{44}	LMN_{45}	…
…	…	…	…	…	…	…

Figure 3-4: LMN411 Pixel Location

When 4:1:1 sub-sampling is used, the components are transmitted using the following order, unless a component order is explicitly stated in the standard referencing the Pixel Format Naming Convention.

L_{11} , L_{12} , M_{11} , L_{13} , L_{14} , N_{11} , L_{15} , L_{16} , M_{15} , L_{17} , L_{18} , N_{15} …

3.1.5 LMNO4444 Location

This format is typically used for any 4 component color space, such as aRGB (where ‘a’ represents alpha compositing). No sub-sampling is performed.

Ex: aRGB8

	1	2	3	4	…
1	$LMNO_{11}$	$LMNO_{12}$	$LMNO_{13}$	$LMNO_{14}$	…
2	$LMNO_{21}$	$LMNO_{22}$	$LMNO_{23}$	$LMNO_{24}$	…
3	$LMNO_{31}$	$LMNO_{32}$	$LMNO_{33}$	$LMNO_{34}$	…
4	$LMNO_{41}$	$LMNO_{42}$	$LMNO_{43}$	$LMNO_{44}$	…
…	…	…	…	…	…

Figure 3-5: LMNO4444 Pixel Location

3.1.6 LM44 Location

This format is typically used for any 2-component pixels, such as Coord3D_AC, used by line scan 3D devices (where B coordinate is implicit or e.g. defined by an encoder position). No sub-sampling is performed.

Ex: Coord3D_AC16

	1	2	3	4	...
1	LM_{11}	LM_{12}	LM_{13}	LM_{14}	...
2	LM_{21}	LM_{22}	LM_{23}	LM_{24}	...
3	LM_{31}	LM_{32}	LM_{33}	LM_{34}	...
4	LM_{41}	LM_{42}	LM_{43}	LM_{44}	...
...	...	...	...	...	...

Figure 3-6 : LM44 Pixel Location

3.1.7 Bayer Location

For Bayer patterns in this section, red = L, green = M and blue = N.

Bayer_LMMN Location

This is the format where the green component occupies the 2nd and 3rd cell within the tile. The red component occupies the first cell while the blue component fills the 4th cell.

Ex: BayerRG8

R	G
G	B

Figure 3-7: BayerRG array

	1	2	3	4	…
1	L_{11}	M_{12}	L_{13}	M_{14}	…
2	M_{21}	N_{22}	M_{23}	N_{24}	…
3	L_{31}	M_{32}	L_{33}	M_{34}	…
4	M_{41}	N_{42}	M_{43}	N_{44}	…
…	…	…	…	…	…

Figure 3-8: Bayer_LMMN Pixel Location

Bayer_NMML Location

This is the format where the green component occupies the 2nd and 3rd cell within the tile. The blue component occupies the first cell while the red component fills the 4th cell.

Ex: BayerBG8

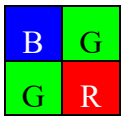


Figure 3-9: BayerBG array

	1	2	3	4	…
1	N_{11}	M_{12}	N_{13}	M_{14}	…
2	M_{21}	L_{22}	M_{23}	L_{24}	…
3	N_{31}	M_{32}	N_{33}	M_{34}	…
4	M_{41}	L_{42}	M_{43}	L_{44}	…
…	…	…	…	…	…

Figure 3-10: Bayer_NMML Pixel Location

Bayer_MLNM Location

This is the format where the green component occupies the 1st and 4th location within the tile. The red component occupies the 2nd cell while the blue component fills the 3rd cell.

Ex: BayerGR8

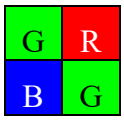


Figure 3-11: BayerGR array

	1	2	3	4	…
1	M_{11}	L_{12}	M_{13}	L_{14}	…
2	N_{21}	M_{22}	N_{23}	M_{24}	…
3	M_{31}	L_{32}	M_{33}	L_{34}	…
4	N_{41}	M_{42}	N_{43}	M_{44}	…
…	…	…	…	…	…

Figure 3-12: Bayer_MLNM Pixel Location

Bayer_MNLM Location

This is the format where the green component occupies the 1st and 4th location within the tile. The blue component occupies the 2nd cell while the red component fills the 3rd cell.

Ex: BayerGB8

G	B
R	G

Figure 3-13: BayerGB array

	1	2	3	4	…
1	M_{11}	N_{12}	M_{13}	N_{14}	…
2	L_{21}	M_{22}	L_{23}	M_{24}	…
3	M_{31}	N_{32}	M_{33}	N_{34}	…
4	L_{41}	M_{42}	L_{43}	M_{44}	…
…	…	…	…	…	…

Figure 3-14: Bayer_MNLM Pixel Location

3.1.8 BiColor_LMNO Location

Bi-color is a color filter array (CFA) that refers to an image composed of two-color component pixels. The sensor can contain up to four color components (L, M, N and O), but each pixel only has information on two of those components (either L and M, or N and O). The missing color components of a pixel can be interpolated from adjacent pixels in a fashion similar to a CFA.

For instance, a two-stage line scan sensor could expose the objects twice with different portions of the sensor, so both portions represent the same physical object location. The color filter array of the sensor could have the red and blue color components on the first line, and the green component on the second line.

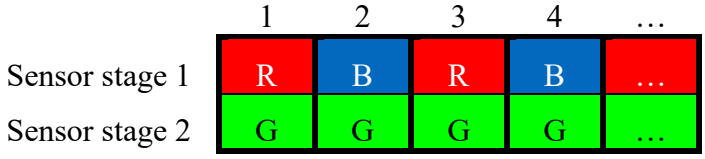


Figure 3-15: Bi-color sensor with 2 stages

The camera combines those 2 stages and output the following data:



Figure 3-16: Bi-color camera output from 2 stages

The above is on example. Other grouping of 2 components into a given pixel location are possible.

The location of the bi-color pixels is represented by the following diagram where 2 components are combined at each location.

	1	2	3	4	…
1	LM_{11}	NO_{12}	LM_{13}	NO_{14}	…
2	LM_{21}	NO_{22}	LM_{23}	NO_{24}	…
…	…	…	…	…	…

Figure 3-17: BiColor_LMNO Pixel Location

3.1.9 Sparse Color Filter Location

Spare Color Filter is a color filter array that includes panchromatic pixels with the red, green and blue color components. Different tile patterns can be created.

For Sparse Color Filter patterns in this section, white = L, blue = M, green = N and red = O.

SCF1_LMLN Location

SCF1_LMLN is a sparse color filter pixel layout where:

1. The panchromatic (white) component occupies the 1st, 3rd, 6th, 8th, 9th, 11th, 14th and 16th location in a 4x4 tile;
2. The green component occupies the 4th, 7th, 10th and 13th cell.
3. The blue component occupies the 2nd and 5th cell.

4. The red component occupies the 12th and 15th cell.

The first line of the tile is thus WBWG.

Ex: SCF1WBWG8

	1	2	3	4	…
1	L_{11}	M_{12}	L_{13}	N_{14}	…
2	M_{21}	L_{22}	N_{23}	L_{24}	…
3	L_{31}	N_{32}	L_{33}	O_{34}	…
4	N_{41}	L_{42}	O_{43}	L_{44}	…
…	…	…	…	…	…

Figure 3-18: SCF1_LMLN Pixel Location

SCF1_LNLM Location

SCF1_LNLM is a sparse color filter pixel layout where:

1. The panchromatic (white) component occupies the 1st, 3rd, 6th, 8th, 9th, 11th, 14th and 16th location in a 4x4 tile;
2. The green component occupies the 2nd, 5th, 12th and 15th cell.
3. The blue component occupies the 4th and 7th cell.
4. The red component occupies the 10th and 13th.

The first line of the tile is thus WGWB.

Ex: SCF1WGWB8

	1	2	3	4	…
1	L_{11}	N_{12}	L_{13}	M_{14}	…
2	N_{21}	L_{22}	M_{23}	L_{24}	…
3	L_{31}	O_{32}	L_{33}	N_{34}	…
4	O_{41}	L_{42}	N_{43}	L_{44}	…
…	…	…	…	…	…

Figure 3-19: SCF1_LNLM Pixel Location

SCF1_LOLN Location

SCF1_LOLN is a sparse color filter pixel layout where:

1. The panchromatic (white) component occupies the 1st, 3rd, 6th, 8th, 9th, 11th, 14th and 16th location in a 4x4 tile;
2. The green component occupies the 4th, 7th, 10th and 13th cell.

3. The blue component occupies the 12th and 15th cell.
4. The red component occupies the 2nd and 5th cell.

The first line of the tile is thus WRWG.

Ex: SCF1WRWG8

	1	2	3	4	…
1	L_{11}	O_{12}	L_{13}	N_{14}	…
2	O_{21}	L_{22}	N_{23}	L_{24}	…
3	L_{31}	N_{32}	L_{33}	M_{34}	…
4	N_{41}	L_{42}	M_{43}	L_{44}	…
…	…	…	…	…	…

Figure 3-20: SCF1_LOLN Pixel Location

SCF1_LNLO Location

SCF1_LNLO is a sparse color filter pixel layout where:

5. The panchromatic (white) component occupies the 1st, 3rd, 6th, 8th, 9th, 11th, 14th and 16th location in a 4x4 tile;
6. The green component occupies the 4th, 7th, 10th and 13th cell.
7. The blue component occupies the 10th and 13th.
8. The red component occupies the 4th and 7th cell.

The first line of the tile is thus WRWG.

Ex: SCF1WGWR8

	1	2	3	4	…
1	L_{11}	N_{12}	L_{13}	O_{14}	…
2	N_{21}	L_{22}	O_{23}	L_{24}	…
3	L_{31}	M_{32}	L_{33}	N_{34}	…
4	M_{41}	L_{42}	N_{43}	L_{44}	…
…	…	…	…	…	…

Figure 3-21: SCF1_LNLO Pixel Location

3.1.10 CFA_xxxx Location (square pattern)

CFA stands for generic “Color Filter Array”. It is used for CFAs other than the popular Bayer tile and Sparse Color Filter defined earlier. ‘xxxx’ explicitly represents the sequence of color components in the square pattern expressed in raster-scan. For example, “CFA_RBGG” would be a CFA pattern with red-blue

on first line and green-only on the second line. It can be used to express CFA larger than 2x2 as illustrated below. For instance “CFA_GWRWGWRWBWGWBWGW” is the sequence represented by the pattern below.

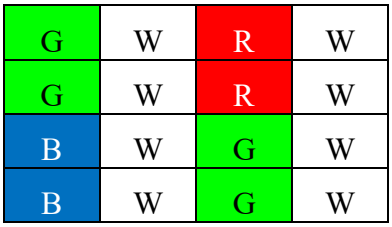


Figure 3-22 : Examples of a generic 4x4 CFA

Note: When a specific CFA pattern becomes widespread, it is possible to assign it a shorter name to reference it. This could be a display name that is more human-readable. To enable interoperability, this short name has to be included in this naming convention. Bayer and Sparse Color Filter are 2 examples used by this convention.

3.1.11 CFA<#lines>by<#columns>_xxxx Location (non-square pattern)

Some Color Filter Arrays (CFA) have a non-square pattern. For these cases, the dimensions of the pattern must be explicitly specified. This is achieved by directly indicating the number of lines followed by the number of columns used by the pattern right after the CFA prefix. The rest of the pixel name follows the same principle of the CFA_xxxx presented above: ‘xxxx’ explicitly represents the sequence of color components in the pattern presented in raster-scan (left to right, then top to bottom). This type of pattern can be used in linescan applications.



Figure 3-23: CFA1by4_GRGB array

3.1.12 Polarized Location

Filters array variants are explicitly listed in this specification to reduce the number of possible pixel formats and thus overall complexity. If new sensors with a different filter array structure are released this filter array definition must be added to this specification. Every new filter array will result in a new subsection in this document.

Naming the formats

Naming is prefixed by POL (for Polarized) followed by the filter kernel size (x and y) followed by the type (M for monochrome or C for color) and a number if there are several different variations possible. The format´s name will be listed in the FPNC-Specification within the following chapters.
In order to allow for a X and/or Y mirroring of the components there can be an added X for flip at X axis and/or a Y for flip at Y axis.

POL22M1: Polarized filter array 2x2 pixels according to POL22M1 below

POL22M1X: As above but filter array is reversed by x.

POL22M1Y: As above but filter array is reversed by y.

POL22M1XY: As above but filter array is reversed by x and y.

POL22M1 Location

POL22M1 defines a 2 by 2 filter array with no color filter applied.

90	45
135	0

Figure 3-24 POL22M1 array

Example of a complete code with 8 bits per pixel:

POL22M1_8: As defined in above figure.

POL22M1XY_8: Reversed by x and y

POL44C1 Location

POL44C1 defines a 4 by 4 filter array with following color filter applied.

90+R	45+R	90+G	45+G
135+R	0+R	135+G	0+G
90+G	45+G	90+B	45+B
135+G	0+G	135+B	0+B

Figure 3-25: POL44C1 array

Example of a complete code with 12 bits per pixel:

POL44C1_12: As defined in above figure.

POL44C1X_12: Reversed by x

3.2 Components

The components provide the primary information of the pixel. Basic component designation might be extended by an indicator providing additional information about pixel positioning in the image (pixel sequence for Bayer, sub-sampling for Y'CbCr, …). When needed, an additional identifier might be inserted to differentiate between 2 very similar formats (such as ITU-R BT.601 and ITU-R BT.709 color space for Y'CbCr).

Table 3-1 : Component Designation

Component designation	Positioning in Image	Description
"Raw"	Mono location	Raw sensor data with no reference to any color space
"Mono"	Mono location	Monochrome (luma only)
"R"	Mono location	Red only
"G"	Mono location	Green only
"B"	Mono location	Blue only
"RGB"	LMN444 location	Red-Green-Blue
"BGR"	LMN444 location	Blue-Green-Red
"BayerGR"	Bayer_MLNM location	Bayer filter Green-Red-Blue-Green

Component designation	Positioning in Image	Description
“BayerRG”	Bayer_LMMN location	Bayer filter Red-Green-Green-Blue
“BayerGB”	Bayer_MNLM location	Bayer filter Green-Blue-Red-Green
“BayerBG”	Bayer_NMML location	Bayer filter Blue-Green-Green-Red
“BiColorRGBG”	BiColor_LMNO location	Bi-color pixel Red/Green - Blue/Green
“BiColorGRGB”	BiColor_LMNO location	Bi-color pixel Green/Red - Green/Blue
“BiColorBGRG”	BiColor_LMNO location	Bi-color pixel Blue/Green - Red/Green
“BiColorGBGR”	BiColor_LMNO location	Bi-color pixel Green/Blue - Green/Red
“aRGB”	LMNO4444 location	alpha-Red-Green-Blue Alpha component should only be used to control transparency for displaying on a display device. 0xff means no transparency.
“YRGB”	LMNO4444 location	Luma-Red-Green-Blue
“CwRGB”	LMNO4444 location	Custom-Red-Green-Blue Custom wavelength can be something like infrared, near-infrared, ultraviolet.
“RGBa”	LMNO4444 location	Red-Green-Blue-alpha Alpha component should only be used to control transparency for displaying on a display device. 0xff means no transparency.
“RGBCw”	LMNO4444 location	Red-Green-Blue-Custom Custom wavelength can be something like infrared, near-infrared, ultraviolet.
“aBGR”	LMNO4444 location	alpha-Blue-Green-Red Alpha component should only be used to control transparency for displaying on a display device. 0xff means no transparency.
“CwBGR”	LMNO4444 location	Custom-Blue-Green-Red Custom wavelength can be something like infrared, near-infrared, ultraviolet.
“BGRa”	LMNO4444 location	Blue-Green-Red-alpha Alpha component should only be used to control transparency for displaying on a display device. 0xff means no transparency.
“BGRCw”	LMNO4444 location	Blue-Green-Red-Custom Custom wavelength can be something like infrared, near-infrared, ultraviolet.

Component designation	Positioning in Image	Description
“YUV” “YUV422” “YUV411”	LMN444 location LMN422 location LMN411 location	YUV color space, typically an incorrect usage of the Y’CbCr color space. Legacy from IIDC standard. *Default*: Y is unsigned, U and V are signed (shifted by adding 128 for 8-bit components)
“YCbCr” “YCbCr422” “YCbCr411”	LMN444 location LMN422 location LMN411 location	Generic Y’CbCr color space using full range of 256 values for each component. See section 8.2.1 for the color transform equations. Y’, Cb and Cr are in the range [0, 255]. Y is unsigned, Cb and Cr are signed (shifted by adding 128).
“YCbCr601” “YCbCr601_422” “YCbCr601_411”	LMN444 location LMN422 location LMN411 location	Y’CbCr color space as specified by ITU-R BT.601 (SDTV). See section 8.2.2 for the color transform equations. Y’ is in the range [16, 235]. Cb and Cr are in the range [16, 240]. Y’ is unsigned, Cb and Cr are signed (shifted by adding 128).
“YCbCr709” “YCbCr709_422” “YCbCr709_411”	LMN444 location LMN422 location LMN411 location	Y’CbCr color space as specified by ITU-R BT.709 (HDTV). See section 8.2.3 for the color transform equations. Y’ is in the range [16, 235]. Cb and Cr are in the range [16, 240]. Y’ is unsigned, Cb and Cr are signed (shifted by adding 128).
“SCF1WBWG”	SCF1_LMLN location	Sparse Color Filter #1, White-Blue-White-Green pattern
“SCF1WGWB”	SCF1_LNLM location	Sparse Color Filter #1, White-Green-White-Blue pattern
“SCF1WRWG”	SCF1_LOLN location	Sparse Color Filter #1, White-Red-White-Green pattern
“SCF1WGWR”	SCF1_LNLO location	Sparse Color Filter #1, White-Green-White-Red pattern
“CIELAB”	LMN444 location	CIE 1976 $L^*a^*b^*$ color space
“CIEXYZ”	LMN444 location	CIE 1931 XYZ color space
“HSI”	LMN444 location	Hue, Saturation, Intensity
“HSV”	LMN444 location	Hue, Saturation, Value
"Coord3D_ABC"	LMN444 location	Used for 3D imaging data. Coordinates of the 3D pixel. The depth/range is always represented by coordinate C. *Note: the coordinate system (meaning and unit of individual coordinates) is defined through other means by each device as specified in the GenICam SFNC standard.*
"Coord3D_A"	Mono location	Used for 3D imaging data. Coordinate A only.
"Coord3D_B"	Mono location	Used for 3D imaging data. Coordinate B only.

Component designation	Positioning in Image	Description
"Coord3D_C"	Mono location	Used for 3D imaging data. Coordinate C only (the coordinate expressing depth/range). *Note: if the C coordinate is transferred alone, the other two coordinates are implicit as specified in the GenICam SFNC standard.*
"Coord3D_AC"	LM44 location	Used for 3D imaging data. Coordinates A and C of the 3D pixel. *Note: intended for line scan 3D devices. The second coordinate is implicit as specified in the GenICam SFNC standard.*
"Confidence"	Mono location	Confidence of the pixel value. Expresses the level of validity of the given pixel value.

Note 1: The full scale R, G or B (256 values) can be replaced by their scaled down version r, g or b (235 values) when necessary.

Unless specified otherwise, the order in which the components are listed is the order they will appear on the wire or in memory. The first component appears in the first byte(s) and so on.

3.2.1 CFA Basic Components

Pixel formats based on a color filter array must explicitly state the basic components used to create the pattern.

Table 3-2 : CFA Basic Components

Basic Component for CFA	Color	Additional Information
"R"	Red	Used in primary color sensor.
"G"	Green	Used in primary and complementary color sensor.
"B"	Blue	Used in primary color sensor.
"W"	White	A pixel with no color filter (panchromatic)
"C"	Cyan	Used in complementary color sensor.
"M"	Magenta	Used in complementary color sensor.
"Ye"	Yellow	Used in complementary color sensor.
"Ir"	Infrared	Used for infrared (IR) channel
"Cw"	Custom wavelength	Used for a color filter other than the traditional RGB or CMY (e.g. infrared, near-infrared, ultraviolet)

3.3 Generic Data Types Formats

This section provides the format naming to be used for generic non-pixel data. Those generic formats should be used only for non-image data block representation. They can be used for example, to specify the format of a metadata block of information, processing results (such as histogram) …

Note that although strictly speaking those formats are not representing pixels, they are defined in this PFNC naming convention in order to keep all data formats used by GenICam in one document.

Table 3-3 : Generic Data Types Designation

Component designation	Positioning	Description
"Data"	Same as Mono location	Generic data.

4 Number of bits for each component

This field provides the number of bits for each component. Typical values are

- 1, 2, 4, 5, 6, 8, 10, 12, 14 and 16 for integer data types;
- 32 and 64 for floating point data types.

Use one number if all components of the pixel have same number of bits (ex: Mono8), otherwise one must successively list one number for each component with no space in-between using the same number of digits for all components (including a leading zero when necessary)

Ex: RGB565 = 5-bit R + 6-bit G + 5-bit B

YCbCr160808 = 16-bit Y' + 8-bit Cb + 8-bit Cr

From the above, one can deduce the number of bits occupied by the pixel (not including padding bits). If a single value is listed, then the number of bits is equal to number of components multiplied by the number of bits:

Ex: RGB8 = 3 components of 8-bit = 24 bits

Coord3D_AC16 = 2 components of 16-bit = 32 bits

When the components don't use the same number of bits, then it is the concatenation of them:

Ex: RGB565 = 5-bit for red + 6-bit for G + 5-bit for B = 16 bits for each pixel

The "Packing Style" section introduces padding bits that increases to overall size of the pixel. In those situations where padding bits are used, the packing style might include a number representing the number of bits used by the pixel, including zero-padding.

5 Optional “Data type” indicator

This field allows to specify the data type of the pixel component values. If omitted, unsigned integer data type is assumed.

Empty or “**u**”	Default for most component. Unsigned integer data.
“**s**”	Signed integer data (two’s complement).
“f”	Floating point data (binary floating point format compatible with IEC 60559:1989 standard) *Note: need to handle specific floating point values, in particular NaN's, during pixel data processing might incur performance penalties, it might be desirable to avoid such values within pixel data whenever possible.*

Use one value if all components have same data type, otherwise must list as many data type indicators as there are components in the pixel, successively with no space in-between in the same order they are presented in the “Components and Location” field.

6 Optional Packing Style

This convention defines optional packing styles and includes an additional tag to align pixel to a certain bit boundary. In most cases, the style needs to support both lsb and msb aligned components. PFNC defines lsb bit ordering as the default for pixel formats received in image buffers.

Important: For image buffers, PFNC defines lsb as the default ordering. Therefore, unless msb is explicitly specified in the pixel name, all pixel formats must use lsb ordering when they are put in a PFNC-compliant image buffer. In this case, the lsb suffix does not need to be spelled out explicitly.

6.1 Unpacked

Unpacked is one of the most prevalent styles where each component occupies an integer number of bytes: padding bits are put as necessary in the least or most significant bits to reach the next 8-bit boundary.

6.1.1 lsb Unpacked

By default, unpacked style uses “lsb unpacked” and does not need to be explicitly specified. When no padding bit is necessary, then “lsb unpacked” designation takes precedence over “msb unpacked”. lsb unpacked is thus the default for 8-bit and 16-bit components.

For lsb unpacked, each component is aligned to the lsb and its msb’s are zero-padded to nearest byte (8-bit) boundary. Hence next component (or pixel) always starts on the next byte. It is the typical pixel format used for image buffers on the PC-side to facilitate image processing.

Note: In the following figures, the ‘p’ stands for padding bit. This means that position is a padding zero.

Note that in the following figures, we put **byte 0 on the right** to help illustrate the concept.

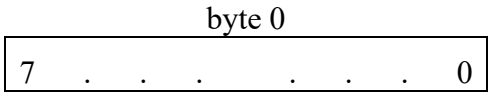


Figure 6-1: Mono8 unpacked

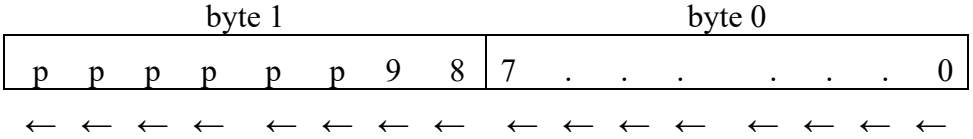


Figure 6-2:Mono10 unpacked

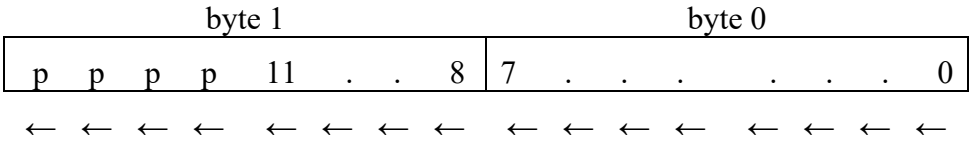


Figure 6-3: Mono12 unpacked

Pixel construction rules for the "lsb Unpacked" style

To construct the pixel stream:
1) Put the component in the least significant bits.
2) Pad the most significant bits to the nearest 8-bit boundary if needed.
3) Start with the next component on the next 8-bit boundary.

6.1.2 msb Unpacked

For msb unpacked, each component is filled msb first and its lsb's are zero-padded to the nearest byte (8-bit) boundary. Hence next component (or pixel) always starts on the next byte. For PFNC-compliant image buffers, msb unpacked must be explicitly specified in the pixel format name by appending "msb".

Note: If the component size is a multiple of 8 bits, then use lsb unpacked since no padding bits is necessary and this convention aims for the shortest string to represent the pixel name.

Note that in the following figures, we put **byte 0 on the left** to help illustrate the concept.

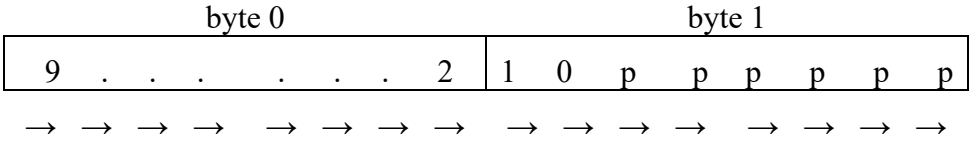


Figure 6-4:Mono10msb unpacked

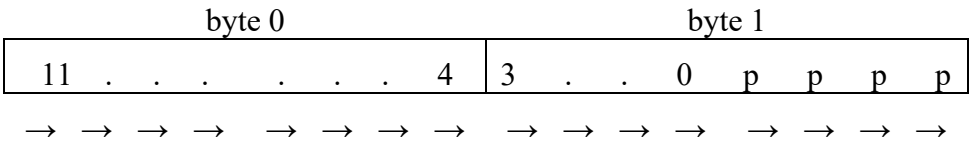


Figure 6-5: Mono12msb unpacked

Pixel construction rules for the “msb Unpacked” style

To construct the pixel stream:
1) Put the component in the most significant bits.
2) Pad the least significant bits to the nearest 8-bit boundary.
3) Start with the next component on the next 8-bit boundary.

6.2 Cluster marker

The cluster marker (“c”) is only allowed for single-component/monochrome pixel formats. It is used to regroup a given number of single-component/monochrome pixels into one multi-component pixel. This facilitates the re-use of some multi-component/color pixel packing style concepts for single-component/monochrome pixels.

The cluster marker is immediately followed by a number indicating the number of single-component/monochrome pixels that are grouped into the cluster.

Ex: c2 = 2 single-component/monochrome pixels in the cluster

c3 = 3 single-component/monochrome pixels in the cluster (which makes the cluster similar to RGB format)

A cluster marker is only required to remove a possible ambiguity with the pixel format name, typically when the number of bits (including padding) is not a multiple of the number of single-component/monochrome pixels in the cluster. In general, the cluster marker should be avoided as it clouds the pixel name and makes it less friendly.

When the cluster marker is used, then the packed or grouped style must consider the cluster of single-component/monochrome pixels as one multi-component pixel. This directly impacts the number immediately following those 2 tags which must now represent the number of bits for the cluster.

The following figure illustrates a scenario where 3 monochrome pixels are regrouped into one 3-component pixel. This 3-component pixel is then lsb packed to 32 bits, leaving 2 padding bits in the msb’s position.

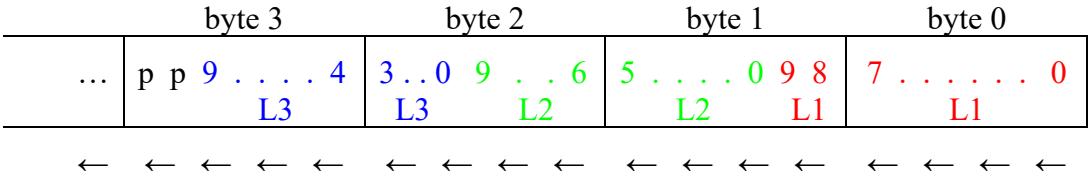


Figure 6-6 : 10-bit monochrome pixel lsb packed into 32 bits (Mono10c3p32)

6.3 Packed tag

Packed (“p”) is a common packing style where there is no bit spacing left between components (or possibly between successive pixels). The packed tag is followed by an optional number providing the number of bits the data is packed into (when it is not using all the available bits and padding bits are necessary) and by an optional bit order indicating if packing starts from lsb or msb. Empty bit must be padded to 0. The first component starts in byte 0.

6.3.1 lsb Packed

lsb packed is the default packed mode and does not need to be explicitly specified after the ‘p’ indicator.

For lsb packed, the data is filled lsb first in the lowest address byte (byte 0) starting with the first component and continue in the lsb of byte 1 (and so on). Padding bits, if any, would thus be the msb’s of the last byte after putting all the components. Padding bits are necessary when the “p” packing tag is followed by a number indicating to how many bits we need to align the pixel.

Note that in the following figures, we put **byte 0 on the right** to help illustrate the concept.

The following figure represents an example of a 3 color component pixel using 10 bits for each color component packed into a 32-bit data. The data is lsb packed; meaning byte 0 contains the least significant bits of the first color component. We start filling data with the lsb of byte 0 and continue with the lsb of byte 1 (and so on). The fact there is a 32 after the “p” packing tag indicates that padding bits are necessary to align the pixel to 32-bit in this example.

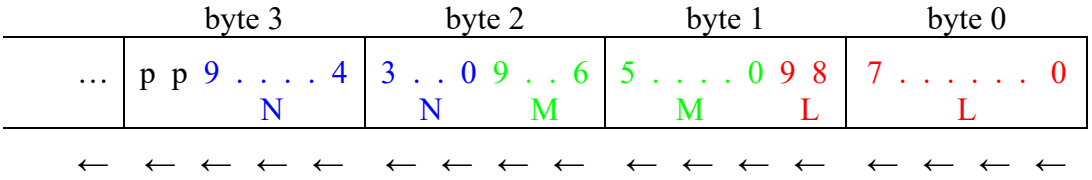


Figure 6-7 :3 components in 10-bit lsb packed into 32-bit pixel (RGB10p32)

Notice that bits are put successively for each component with no spacing in-between, followed by the padding bits.

Here is another example typical for RGB565 lsb packed. Notice there is no number after the “p” packing tag hence no padding bit.

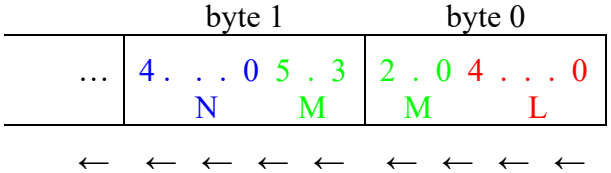


Figure 6-8 :3 components lsb packed into 16-bit pixel (RGB565p)

The following example shows how a 10-bit monochrome data can be packed from its lsb, again with no padding bit.

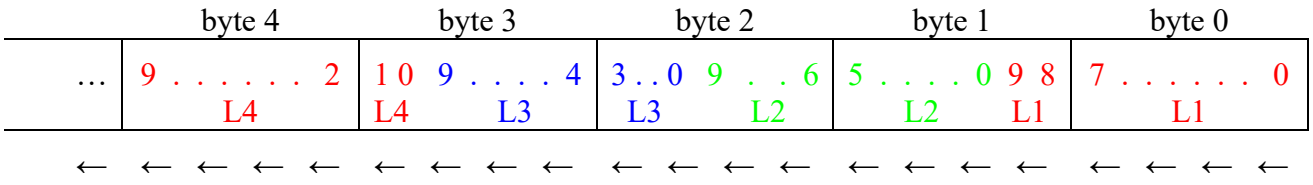


Figure 6-9 : 10-bit monochrome pixel lsb packed (Mono10p)

Pixel construction rules for the "lsb Packed" style

The total number of bits after packing is either:
1) Indicated by the number following the "p" tag when present
2) Deduced by putting as many components such that no padding bit is required.

To construct the pixel stream:
1) Take the first component and put it in the lsb's of the first byte, with bit 0 holding the lsb of the component. Extra bits of this component continue in the lsb's of the next byte.
2) Then take the following component and append it to the first one, again starting from the lsb of the component.
3) Proceed in this way, appending the next component from its lsb, until no more components left.
4) Pad the last byte's most significant bits with 0 if needed (i.e. to meet the total number of bits indicated after the "p" tag). This padding must consider the line or image boundary, as explained in section 6.7.

6.3.2 msb Packed

For msb packed, the data is filled msb first in the lowest address byte (byte 0), starting with the first component. For PFNC-compliant image buffers, msb packed must be explicitly specified in the pixel format name by appending "msb" after the 'p' (i.e. "pmsb").

Note that in the following figure, we put **byte 0 on the left** to help illustrate the concept. The data is filled msb first in the lowest address byte (byte 0) starting with the first component and continue in the msb of byte 1 (and so on). Padding bits, if any, would thus be the lsb's of the last byte after putting all the components. Padding bits are necessary when the "p" packing tag is followed by a number indicating to how many bits we need to align the pixel.

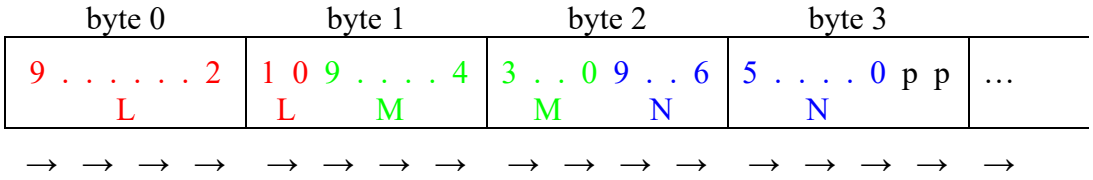


Figure 6-10 :3 components in 10-bit msb packed into 32-bit pixel (RGB10p32msb)

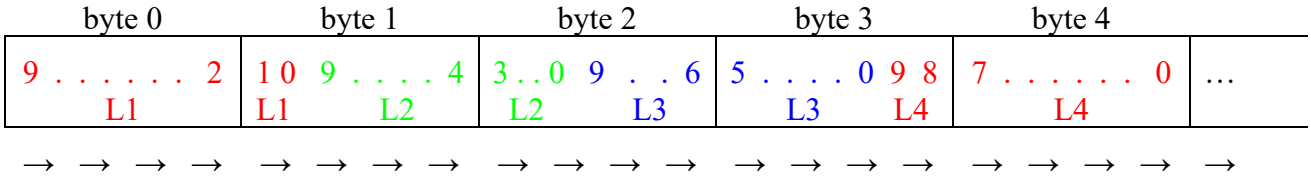


Figure 6-11 : 10-bit monochrome pixel msb packed (Mono10pmsb)

Pixel construction rules for the "msb Packed" style

The total number of bits after packing is either:
1) Indicated by the number following the "p" tag when present; or
2) Deduced by putting as many components such that no padding bit is required.

To construct the pixel stream:
1) Take the first component and put it in the msb's of the first byte, with bit 7 holding the msb of the component. Extra bits of this component continue in the msb's of the next byte.
2) Then take the following component and append it to the first one, again starting from the msb of the component.
3) Proceed in this way, appending the next component from its msb, until no more components left.
4) Pad the last byte's least significant bits with 0 if needed (i.e. to meet the total number of bits indicated after the "p" tag). This padding must consider the line or image boundary, as explained in section 6.7.

6.4 Grouped tag

Grouped ("g") is a different packing style created by regrouping extra lsb's or msb's of components (or from successive pixels) in a separate byte(s). The format indicates the number of bits the data occupies when it is different than the nominal bits per pixel for the given component (i.e. including the padding bits). ex: g12 when grouped into 12 bits. This is followed by an optional grouping order indicating if the byte containing the extra data is the lsb's or msb's. Empty bit must be padded with 0. The first component is put in byte 0, second component in byte 1 and so on.

When grouped style is used, the byte holding the grouped data shall be put as the last byte(s).

6.4.1 lsb Grouped

lsb grouped is the default grouping mode and does not need to be explicitly specified after the 'g' indicator.

For lsb grouped, the msb's of the components are extracted and put in sequence starting with the first component in byte 0. The lsb's of the components are grouped together in a separate byte that is put last. This last byte is filled by grouping components starting from the lsb using the component order.

Note that in the following figures, we put **byte 0 on the right** to help illustrate the concept.

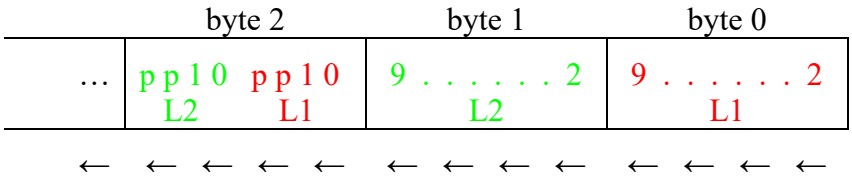


Figure 6-12: 2 monochrome 10-bit pixels with lsb grouped into 12 bits (Mono10g12)

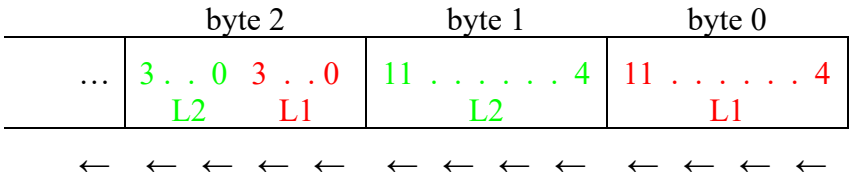


Figure 6-13: 2 monochrome 12-bit pixels with lsb grouped into 24 bits (Mono12g)

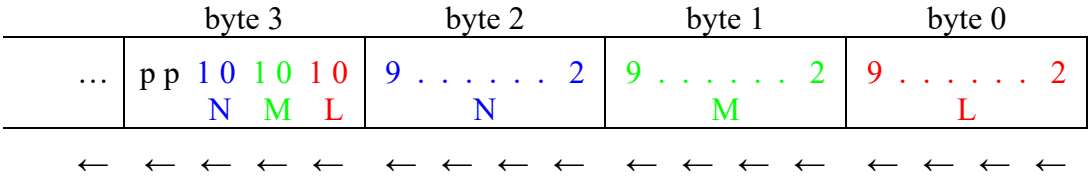


Figure 6-14: 3 components of 10-bit with lsb grouped into 32-bit pixel (RGB10g32)

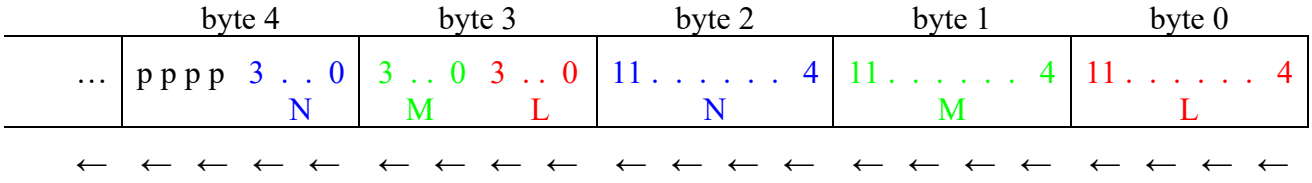


Figure 6-15 : 3 components of 12-bit with lsb grouped into 40-bit pixel (RGB12g40)

Pixel construction rules for the "lsb Grouped" style

This packing style is applicable only when each component contains more than 8 bits but no more than 12 bits.

To construct the pixel stream:
1) Take the 8 msb's of the first component and put them in the first byte. Reserve the extra lsb's of this component for the last byte(s).
2) Take the 8 msb's of the second component and put them in the second byte. Reserve the extra lsb's of this component for the last byte(s).
3) Proceed in this way, taking the 8 msb's of the next component and putting it in the next byte until no more components left. For each component, reserve the extra lsb's of this component for the last byte(s). This grouping could stop at the line or image boundary, as explained in section 6.7.
4) Start filling the last byte(s) from its lsb by successively using the extra lsb's from the first component. For monochrome components, add msb padding bits next to the component extra lsb's such that it occupies the indicated number of bits for the monochrome pixel before proceeding with the next component. Continue filling the last byte(s) using the previous rule for each component in turn.
5) Pad the last byte's msb's with 0 if needed.

6.4.2 msb Grouped

For msb grouped, the lsb's of the components are extracted and put in sequence starting with the first component in byte 0. The msb's of the components are grouped together in a separate byte that is put last. The principle is the same as lsb grouped. The last byte is filled by grouping components starting from the lsb using the component order, with no empty bit in between. For PFNC-compliant image buffers, msb grouped must be explicitly specified in the pixel format name by appending "msb".

Note that in the following figure, we put **byte 0 on the left** to help illustrate the concept.

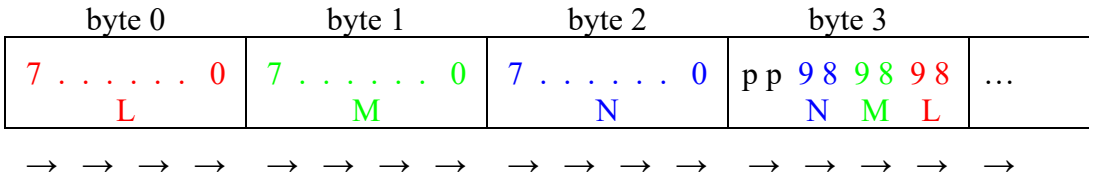


Figure 6-16 : 3 components of 10-bit with msb grouped into 32-bit pixel (RGB10g32msb)

Pixel construction rules for the "msb Grouped" style

This packing style is applicable only when each component contains more than 8 bits but no more than 12 bits.

To construct the pixel stream:
1) Take the 8 lsb's of the first component and put them in the first byte. Reserve the extra msb's of this component for the last byte(s).
2) Take the 8 lsb's of the second component and put them in the second byte. Reserve the extra msb's of this component for the last byte(s).
3) Proceed in this way, taking the 8 lsb's of the next component and putting it in the next byte until no more components left. For each component, reserve the extra msb's of this component for the last byte(s). This grouping could stop at the line or image boundary, as explained in section 6.7.
4) Start filling the last byte(s) from its lsb by successively using the extra msb's from the first component. For monochrome components, add msb padding bits next to the component extra msb's such that it occupies the indicated number of bits for the monochrome pixel before proceeding with the next component. Continue filling the last byte(s) using the previous rule for each component in turn.
5) Pad the last byte's msb's with 0 if needed.

6.5 Align tag

Align ("a") tag can be used to complement the packed and grouped styles. It indicates the total number of bits to align the pixel (if the packing style refers to multi-components) or cluster (if the packing style refers to packing of single-component/monochrome pixels) when there is at least one full byte of padding zeros.

Alignment bits must be set to 0 (they are padding bits). The alignment bytes must be put after any bytes containing component information.

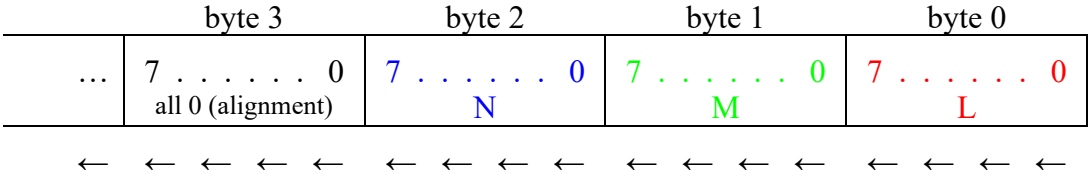


Figure 6-17: RGB 8-bit unpacked aligned to 32-bit (RGB8a32)

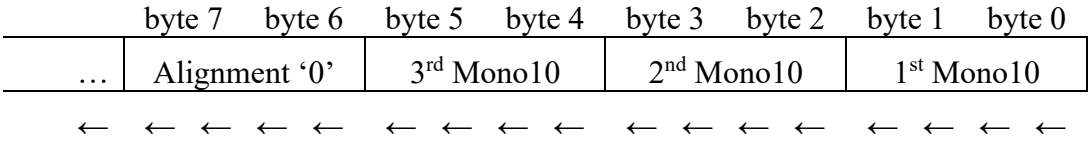


Figure 6-18 : Using a cluster marker of 3 unpacked Mono10 aligned to 64 bits (Mono10c3a64)

Pixel construction rules for the “Align” style

This packing style is applicable only when at least one full byte contains padding bits and alignment must be on an 8-bit boundary.

To construct the pixel stream:
1) Use the unpacked, packed or grouped style specified by the pixel format using the pixel construction rules above.
2) Pad the most significant bits with as many padding bits required to align the data to the number of bits specified by the ‘a’ tag.

6.6 Packing Style Summary

Table 6-1 : Packing Style Summary

Unpacked	Most significant bits of each component are padded with zeros to be 8-bit aligned.
Unpacked msb	Least significant bits of each component are padded with zeros to be 8-bit aligned.
"c<x>"	Cluster of 'x' single-component/monochrome pixels to be regrouped and consider as one multi-component pixel. This marker must appear before the other tags. Used in conjunction with another packing style (packed, grouped or aligned).
"p<x>"	lsb packing on 'x' bits. Components are packed with no bit spacing, lsb's in the first byte. Byte 0's lsb contains the lsb of the first component. Next component is appended going from lsb to msb. Padding bits, if any, occupies the msb's. If packing style does not explicitly include lsb or msb, then lsb is assumed. 'x' indicates the number of bits consumed by the pixel, including the padding 0. 'x' is not necessary when there is no padding bit in the resulting pixel data word.
"p<x>msb"	msb packing on 'x' bits Components are packed with no bit spacing, msb's in the first byte. Byte 0's msb contains the msb of the first component. Next component is appended going from msb to lsb. Padding bits, if any, occupies the lsb's. 'x' indicates the number of bits consumed by the pixel, including the padding 0. 'x' is not necessary when there is no padding bit in the resulting pixel data word.
"g<*x*>"	lsb grouping style on 'x' bits Least significant bits of the components are grouped together in a separate byte(s). If packing style does not explicitly include lsb or msb, then lsb is assumed. Byte 0 contains the first component. 'x' indicates the number of bits consumed by the pixel, including the padding 0. 'x' is not necessary when there is no padding bit in the resulting pixel data word.
"g<*x*>msb"	msb grouping style on 'x' bits Most significant bits of the components are grouped together in a separate byte(s). Byte 0 contains the first component. 'x' indicates the number of bits consumed by the pixel, including the padding 0. 'x' is not necessary when there is no padding bit in the resulting pixel data word.
"*a<x>*"	The pixel (or group of pixels) is aligned to the given bit-boundary. This can be used to complement unpacked, packed or grouped packing style. This tag must appear after any other packing style tags. 'x' indicates the total number of bits to use for this grouping. Unused bit are set to 0. For multiple single-component/monochrome pixels that must be aligned together, it is **mandatory** to use the cluster marker ('c').

6.7 Dealing with Line and Image Boundaries

The packing styles in this section do not specify how it is affected at a line or image boundary. Two options are allowed when data is put in a PFNC-compliant image buffer:

1. **Image Padding**: no artificial padding is inserted at the end of a line. Hence the first pixel of a given line might not start on an 8-bit boundary as it might be combined in the same byte as the last pixel of the previous line. At the end of the image, missing luma components from a cluster take a value of 0
2. **Line Padding**: the last pixel of each line is padded to complete on an 8-bit boundary (or to the boundary specified by the standard referencing this convention), so the first pixel of the next line starts on a fresh 8-bit boundary. At the end of the line, missing luma components from a cluster take a value of 0

Important: The standard referencing this Pixel Format Naming Convention is expected to explicitly define in their document the method used when dealing with line and image boundaries for data put in a PFNC-compliant image buffer. A special treatment might be necessary for 3D data types.

As an example, assume a Mono1p image where the image width is not a multiple of 8. The last pixel of the first line does not align to an 8-bit boundary and a choice must be made between image padding, where pixels from different lines might be packed/grouped together, and line padding where pixels from different lines are not packed/grouped together.

Note: The last pixel at the image or line boundary might need special considerations when grouped or packed style is used. If a component is missing to complete the packing group, then one or more additional 'artificial' components with a value of 0 must be used.

For instance, assume the pixel format uses a cluster of 3 luma components with line padding, but only 2 are left at the end of the line. In this case, an extra luma with a value of 0 must be used to complete this cluster to enable the packing.

Note: Special care might be required when working with pixel data which are not line oriented (such as a 3D point cloud). In such case line padding might not apply. If there could be a risk for any confusion, the camera might elect to prefer pixel formats that ensure each pixel starts at an 8-bit boundary.

7 Interface-specific

The various camera interfaces in the machine vision industry might need to provide additional information about how the data is put onto the interface. This might include the sequence of components in the data packets or the usage of multiple streams to transfer the various components (ex: planar mode).

Any interface-specific field must be delimited by an underscore '_' from the rest of the pixel text.

Although this convention does not try to cover all possibilities, it does list the following typical use cases.

7.1 Planar mode

When each component is transmitted over a different stream/plane or as a separate single-component/monochromatic image, then the pixel format should use the "Planar" suffix.

Ex: RGB10_Planar transmitted as 3 different streams: red plane, green plane and blue plane

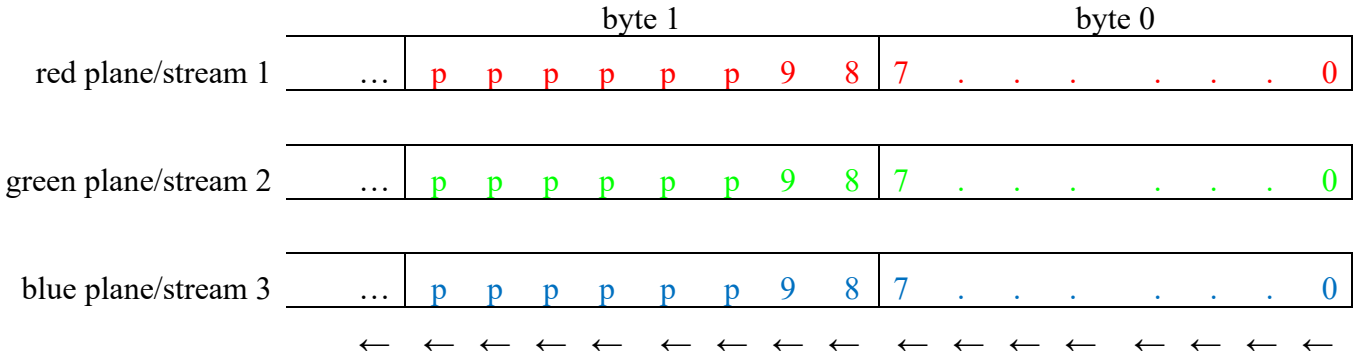


Figure 7-1: RGB10_Planar

The camera interface standard must indicate how those streams are transmitted.

7.2 Semiplanar mode

When two or more components are combined in one stream/plane, then the pixel format should use the "Semiplanar" suffix. Hence within this pixel format we have 1 or more planes with multiple components.

Ex: YCbCr420 Semi-Planar mode as 2 different streams: one Y plane and one CbCr plane with interleaved and sub-sampled blue and red data.

Plane/stream1	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	…
Plane/stream2	Cb0	Cr0	Cb2	Cr2	Cb4	Cr4	Cb6	Cr6	…

To symbolize the plane/component sequence in a Semiplanar pixel format name, an underscore ('_') is used to separate the planes.

Ex: YCbCr420_8_YY_CbCr_Semiplanar . Here we have a plane/component sequence YY_CbCr with 2 planes separated by '_', one YY plane and one CbCr plane.

7.3 Components Sequencing

If the component sequence is not identical to the one specified in the "Components and Location" field, then the actual sequence should be provided by listing as many components as necessary to correctly determine the correct sequence in image memory.

For instance, there are various sequences of components for Y'CbCr, some sending Y', followed by Cb and Cr. But there are others where Cr is put first. This is further complicated by sub-sampling of the chroma components. In these cases, it might be necessary to unequivocally list the order that the components are transmitted.

In this case, a underscore ('_') is put before listing the sequence of component to clearly separate the list from the rest of the pixel name.

Ex: YUV411_8_UYYVYY

8 Appendix A - Color Space Transforms

This section describes the equations a camera should use to convert from gamma-corrected R'G'B' color space into a new color space. The "prime" symbol denotes a gamma corrected value. Most equations assume 8-bit color components (ranging from 0 to 255) and can be easily adjusted for different bit depths.

8.1 Gamma Correction

Gamma correction is used to compensate for non-linearity of the display apparatus. This is a non-linear operation that might impact digital image processing. For instance, a threshold is no longer linear. But it can be useful to amplify low-intensity details at the expense of brighter image details.

$R' = R^{1/\gamma}$

$G' = G^{1/\gamma}$

$B' = B^{1/\gamma}$

where γ is the gamma value used for the correction

Equation 1 : Gamma Correction

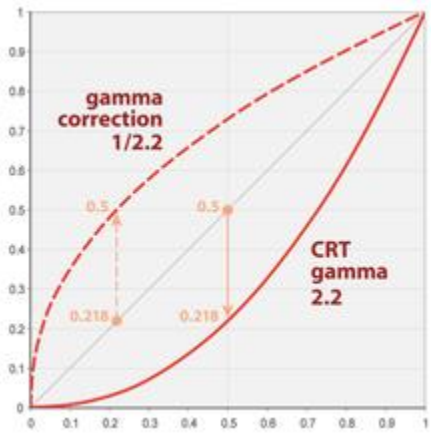


Figure 8-1: Gamma Correction for ITU-R BT.601
(image from Wikipedia)

The prime symbol (') is used to indicate a gamma-corrected component. In the literature, the prime symbol is frequently omitted creating some confusion.

8.2 Y'CbCr Conversions

Many variants of Y'CbCr exist. Most of them are derived from 2 ITU-R specifications:

1. BT.601 (Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios) used for standard television
2. BT.709 (Parameter values for the HDTV standards for production and international programme exchange) used for high definition television

Conversions are typically performed starting from the RGB format. Some of the complexity comes from the range of values permitted for each color component. The computer world typically uses full scale value (i.e. 256 values in 8-bit), while BT.601 and BT.709 use a scaled down range spanning 220 values for Y' and 225 values for Cb and Cr.

BT.601 defines the following basic equation for luma in the analog domain:

$$E'_Y = 0.299\,E'_R + 0.587\,E'_G + 0.114\,E'_B \quad (1)$$

BT.709 defines the following basic equation for luma in the analog domain:

$$E'_Y = 0.2126\,E'_R + 0.7152\,E'_G + 0.0722\,E'_B \quad (2)$$

In the above equations, E'_Y, E'_R, E'_G and E'_B can take floating point value spawning the range [0.0, 1.0].

E'_Y represents the luma information (gamma-corrected luminance). Two color difference components are derived from E'_Y:

$$E'_B - E'_Y \quad (3)$$

$$E'_R - E'_Y$$

To facilitate the notation, this convention defines R', G' and B' as the gamma-corrected full scale values of the RGB color components, while their counterpart r', g' and b' are the gamma-corrected scaled down values as per BT.601 and BT.709. This section provides equations for 8-bit per color component, but a similar reasoning can be established for 10-bit components (or other bit depths).

R', G' and B' in the range [0, 255] (8-bit) (4)

r', g' and b' in the range [16, 235] (8-bit)

8.2.1 Generic Full Scale Y'CbCr (8-bit)

The full scale Y'CbCr is derived from using the basic luma equation from BT.601 and by having Y', Cb and Cr occupy the full 8-bit range of possible values. This format is not covered by BT.601 or BT.709, but often used in computer systems.

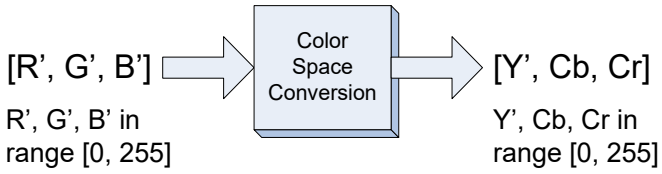


Figure 8-2 : Generic full scale Y'CbCr

From the coefficients in (1), one can determine the possible range of values for the 2 color difference signals shown in (3):

$E'_B - E'_Y$ is in the range [-0.886, +0.886] (5)

$E'_R - E'_Y$ is in the range [-0.701, +0.701]

The next step is to normalize the 2 color difference so they occupy the full scale of [-0.5, +0.5].

$E'_{Cb} = (0.5 / 0.886)(E'_B - E'_Y)$ (6)

$E'_{Cr} = (0.5 / 0.701)(E'_R - E'_Y)$

where E'_{Cb} and E'_{Cr} are in the range [-0.5, +0.5].

In 8 bits, Y', Cb and Cr are derived by normalizing E'_Y, E'_{Cb} and E'_{Cr} to [0, 255]. Note that Cb and Cr are signed shifted by 128 since E'_{Cb} and E'_{Cr} are in the range [-0.5, 0.5]. Including (6) leads to:

$Y' = 255\ E'_Y$ (7)

$Cb = 255\ E'_{Cb} + 128 = 255 \times (0.5 / 0.886)(E'_B - E'_Y) + 128$

$Cr = 255\ E'_{Cr} + 128 = 255 \times (0.5 / 0.701)(E'_R - E'_Y) + 128$

Full scale R', G' and B' in 8 bits offers the following equations:

$R' = 255\ E'_R$ (8)

$G' = 255\ E'_G$

$B' = 255\ E'_B$

Replacing (8) into (7) leads to:

Y' =	0.299 R'	+ 0.587 G'	+ 0.114 B'	
Cb =	-0.16874 R'	- 0.33126 G'	+ 0.5000 B'	+ 128
Cr =	0.5000 R'	- 0.41869 G'	- 0.08131 B'	+ 128

with R', G' and B' in the range [0, 255].

Equation 2 : Generic full scale R'G'B' to Y'CbCr conversion (8 bits)

For 8-bit data, the valid range of values for each component is:

- Y', R', G' and B' in range [0, 255], unsigned (256 levels)
- Cb and Cr in range [0, 255], signed shifted by 128, with 128 representing 0 (256 levels)

Values must be truncated to fit in that range.

Note that the above equations are the ones specified by the JFIF specification (JPEG File Interchange Format).

The reverse equations are given by:

R' =	Y'		+ 1.40200 (Cr – 128)
G' =	Y'	- 0.34414 (Cb – 128)	- 0.71414 (Cr – 128)
B' =	Y'	+ 1.77200 (Cb – 128)	

with Y', Cb and Cr in the range [0, 255].

Equation 3 : Generic full scale Y'CbCr to R'G'B' conversion (8 bits)

Equivalently, the same set of equations can be used for generic YUV where the range of values for each component must use the full 8-bit. In this case, U = Cb and V = Cr.

8.2.2 Y'CbCr601 (8-bit)

ITU-R BT.601 provides a definition of the Y', Cb and Cr based on (1). It defines the following signal range:

Y' in the range [16, 235] (9)

Cb and Cr in the range [16, 240]

Since BT.601 is based on (1), it leads to the same color difference signal indicated in (5) and (6).

Considering (9), we need to normalize (6) into 8-bit components that do not occupy the full 256 values:

$$Y'_{601} = 219\,E'_Y + 16 \qquad (10)$$

$$Cb = 224\,E'_{Cb} + 128 = 224 \times (0.5 / 0.886)\,(E'_B - E'_Y) + 128$$

$$Cr = 224\,E'_{Cr} + 128 = 224 \times (0.5 / 0.701)\,(E'_R - E'_Y) + 128$$

At this point, two options exist depending on the allowed range for RGB components. This does not create a different pixel format for Y'CbCr601, but mainly determines two set of equations depending on the input range of values used for the RGB component.

Full scale RGB

Full scale RGB uses (8) to define the relationship between the R', G' and B' components and E'_R, E'_G and E'_B.

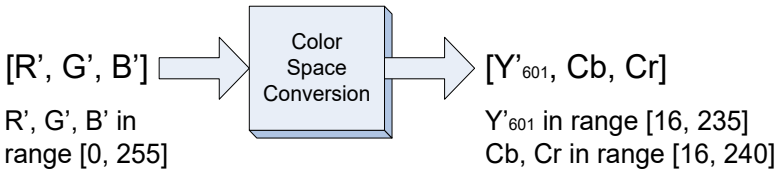


Figure 8-3 : Full scale RGB for BT.601

Replacing (8) into (10) leads to:

$$Y'_{601} = 0.25679\,R' + 0.50413\,G' + 0.09791\,B' + 16$$

$$Cb = -0.14822\,R' - 0.29099\,G' + 0.43922\,B' + 128$$

$$Cr = 0.43922\,R' - 0.36779\,G' - 0.07143\,B' + 128$$

with R', G' and B' in the range [0, 255].

Equation 4 : Full scale R'G'B' to Y'CbCr601 conversion (8 bits)

For 8-bit data, the valid range of values for each component is:

- R', G' and B' in range [0, 255], unsigned (256 levels)
- Y'_{601} in the range [16, 235], unsigned (220 levels)
- Cb and Cr in range [16, 240], signed shifted by 128, with 128 representing 0 (225 levels)

Values must be truncated to fit in that range.

The reverse equations are given by:

$$
\begin{aligned}
R' &= 1.16438\,(Y'_{601} - 16) + 1.59603\,(Cr - 128)\\
G' &= 1.16438\,(Y'_{601} - 16) - 0.39176\,(Cb - 128) - 0.81297\,(Cr - 128)\\
B' &= 1.16438\,(Y'_{601} - 16) + 2.01723\,(Cb - 128)
\end{aligned}
$$

with Y'_{601} in the range [16, 235] and, Cb and Cr in the range [16, 240].

Equation 5 : Full scale Y'CbCr601 to R'G'B' conversion (8 bits)

Scaled down rgb

BT.601 indicates that the RGB components can use a reduced range of values of [16, 235].

Note: The Pixel Format Naming Convention does not define any RGB pixel format using that range of values. But the color conversion equations are provided for completeness since they are referenced by BT.601.

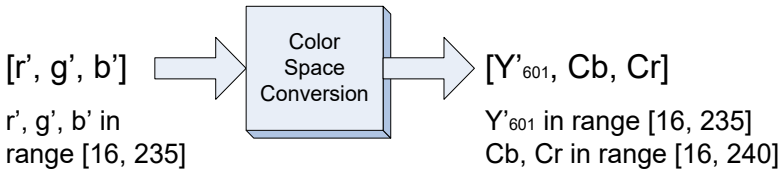


Figure 8-4 : Scaled down rgb for BT.601

This leads to the following equations:

$$
\begin{aligned}
r' &= 219\,E'_R + 16 \qquad (11)\\
g' &= 219\,E'_G + 16\\
b' &= 219\,E'_B + 16
\end{aligned}
$$

where r', g' and b' are in the range [16, 235]

Replacing (11) in (10) leads to:

$$
\begin{aligned}
Y'_{601} &= 0.299\,r' + 0.587\,g' + 0.114\,b'\\
Cb &= -0.17259\,r' - 0.33883\,g' + 0.51142\,b' + 128\\
Cr &= 0.51142\,r' - 0.42825\,g' - 0.08317\,b' + 128
\end{aligned}
$$

with r', g' and b' in the range [16, 235].

Equation 6 : Scaled down r'g'b' to Y'CbCr601 conversion (8 bits)

For 8-bit data, the range of values for each component is:

- Y'_{601}, r', g' and b' in range [16, 235], unsigned (220 levels)
- Cb and Cr in range [16, 240], signed shifted by 128, with 128 representing 0 (225 levels)

Values must be truncated to fit in that range.

The reverse equations are given by:

$$\begin{aligned} r' &= Y'_{601} + 1.37071\,(Cr - 128) \\ g' &= Y'_{601} - 0.33645\,(Cb - 128) - 0.69820\,(Cr - 128) \\ b' &= Y'_{601} + 1.73245\,(Cb - 128) \end{aligned}$$

with Y'_{601} in the range [16, 235] and, Cb and Cr in the range [16, 240].

Equation 7 : Y'CbCr601 to r'g'b' conversion (8 bits)

8.2.3 Y'CbCr709 (8-bit)

ITU-R BT.709 provides a definition of the Y', Cb and Cr based on (1). It defines signal range identical to BT.601, as expressed in (9).

But its luma equation is based on (2). Hence its 2 color difference signals are given by:

$$E'_B - E'_Y \quad \text{is in the range } [-0.9278, +0.9278] \qquad (12)$$
$$E'_R - E'_Y \quad \text{is in the range } [-0.7874, +0.7874]$$

After normalization to occupy the [-0.5, 0.5] range:

$$E'_{Cb} = (0.5 / 0.9278)\,(E'_B - E'_Y) \qquad (13)$$
$$E'_{Cr} = (0.5 / 0.7874)\,(E'_R - E'_Y)$$

Considering (9) that provides the range for Y', Cb and Cr, (13) leads to:

$$Y'_{709} = 219\,E'_Y + 16 \qquad (14)$$
$$Cb = 224\,E'_{Cb} + 128 = 224 \times (0.5 / 0.9278)\,(E'_B - E'_Y) + 128$$
$$Cr = 224\,E'_{Cr} + 128 = 224 \times (0.5 / 0.7874)\,(E'_R - E'_Y) + 128$$

Again, two options exist depending on the allowed range of values for the RGB components.

Full scale RGB

Full scale RGB uses (8) to define the relationship between the R', G' and B' components and E'_R, E'_G and E'_B.

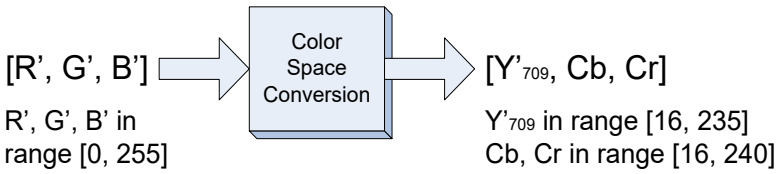


Figure 8-5 : Full scale RGB for BT.709

Replacing (8) into (14) leads to:

$$
\begin{aligned}
Y'_{709} &= 0.18259\,R' + 0.61423\,G' + 0.06201\,B' + 16 \\
Cb &= -0.10064\,R' - 0.33857\,G' + 0.43922\,B' + 128 \\
Cr &= 0.43922\,R' - 0.39894\,G' - 0.04027\,B' + 128
\end{aligned}
$$

with R', G' and B' in the range [0, 255].

Equation 8 : Full scale R'G'B' to Y'CbCr709 conversion (8 bits)

For 8-bit data, the valid range of values for each component is:

- R', G' and B' in range [0, 255], unsigned (256 levels)
- Y'_{709} in the range [16, 235], unsigned (220 levels)
- Cb and Cr in range [16, 240], signed shifted by 128, with 128 representing 0 (225 levels)

Values must be truncated to fit in that range.

The reverse equations are given by:

$$
\begin{aligned}
R' &= 1.16438\,(Y'_{709} - 16) + 1.79274\,(Cr - 128) \\
G' &= 1.16438\,(Y'_{709} - 16) - 0.21325\,(Cb - 128) - 0.53291\,(Cr - 128) \\
B' &= 1.16438\,(Y'_{709} - 16) + 2.11240\,(Cb - 128)
\end{aligned}
$$

with Y'_{709} in the range [16, 235] and, Cb and Cr in the range [16, 240].

Equation 9 : Full scale Y'CbCr601 to R'G'B' conversion (8 bits)

Scaled down rgb

BT.709 indicates that the RGB components can use a reduced range of values of [16, 235]. This corresponds to the equations (11).

Note: The Pixel Format Naming Convention does not define any RGB pixel format using that range of values. But the color conversion equations are provided for completeness since they are referenced by BT.709.

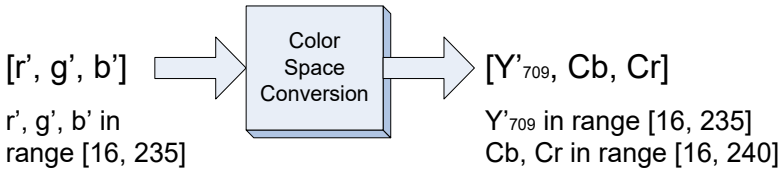


Figure 8-6 : Scaled down rgb for BT.709

Replacing (11) in (14) leads to:

$$
\begin{aligned}
Y'_{709} &= 0.2126\,r' + 0.7152\,g' + 0.0722\,b' \\
Cb &= -0.11719\,r' - 0.39423\,g' + 0.51142\,b' + 128 \\
Cr &= 0.51142\,r' - 0.46452\,g' - 0.04689\,b' + 128
\end{aligned}
$$

with r', g' and b' in the range [16, 235].

Equation 10 : Scaled down r'g'b' to Y'CbCr709 conversion (8 bits)

For 8-bit data, the range of values for each component is:

- Y'_{709}, r', g' and b' in range [16, 235], unsigned (220 levels)
- Cb and Cr in range [16, 240], signed shifted by 128, with 128 representing 0 (225 levels)

Values must be truncated to fit in that range.

The reverse equations are given by:

$$
\begin{aligned}
r' &= Y'_{709} + 1.53965\,(Cr - 128) \\
g' &= Y'_{709} - 0.18314\,(Cb - 128) - 0.45768\,(Cr - 128) \\
b' &= Y'_{709} + 1.81418\,(Cb - 128)
\end{aligned}
$$

with Y'_{709} in the range [16, 235] and, Cb and Cr in the range [16, 240].

Equation 11 : Y'CbCr709 to R'G'B' conversion (8 bits)

9 Appendix B - Sub-sampling notation

The standard sub-sampling notation uses J:a:b convention, where J represents the number of horizontal pixels on a given reference block (the block is always 2 pixels high). Typically, J = 4 and the reference block is 4 pixel wide by 2 lines (illustrated in green in the figure below). The indicator “a” provides the number of chroma samples on the first line while indicator “b” is the number of chroma samples on the second line of the reference block. Luma is not sub-sampled. Both the red and blue chroma are in the same ratio compared to luma.

The position of the chroma samples in relation to the luma can be in two forms:

1. Co-sited
2. Centered

Current version of this convention assumes co-sited positioning, unless noted differently.

Note: This convention could be used for other color components than chroma although common usage is currently limited to YUV and YCbCr.

9.1 Co-sited Positioning

With co-sited positioning, the chroma samples are aligned with the first luma sample of the reference block. Figure 9-1 uses co-sited alignment where the first chroma sample (represented by a black dot) is centered in the upper-left pixel of the image. This is the default chroma sample alignment used by this pixel format naming convention.

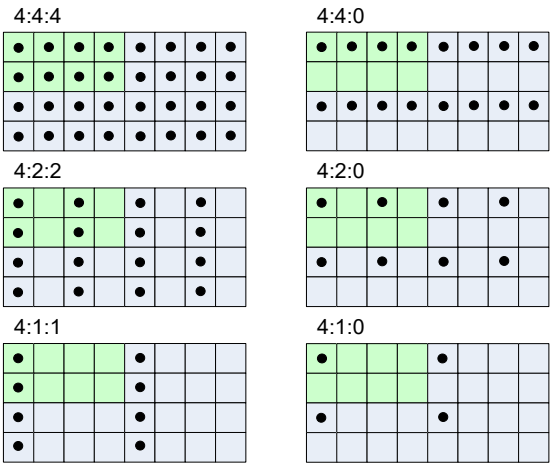


Figure 9-1 : Chroma positioning (co-sited alignment)

ITU-R BT.601 and ITU-R BT.709 require the chroma samples to be co-sited with luma samples (i.e. the first active chroma samples must be co-sited with the first active luma sample).

9.2 Centered Positioning

When centered positioning is used, the chroma samples are put mid-way between the chroma samples that have been averaged during the decimation process.

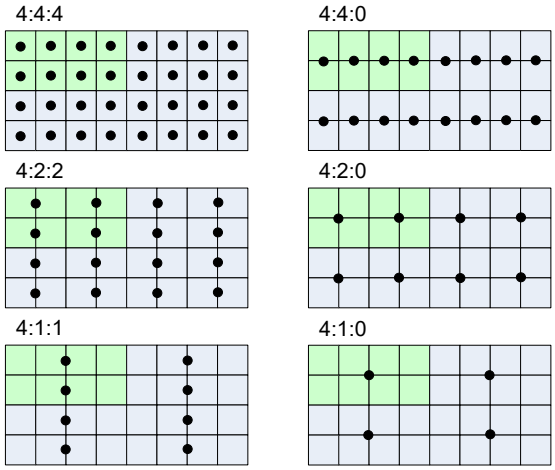


Figure 9-2 : Chroma positioning (centered alignment)

The TIFF file format uses centered chroma sample positioning by default.

10 Appendix C - Pixel Format Value Reference

The GenICam Standard Feature Naming Convention (SFNC) defines a PixelFormat feature that is typically mandatory for Machine Vision camera standards using GenICam.

There are 32 bit identifiers assigned to these standardized pixel formats, which are maintained as part of the PFNC. These 32 bit values are used to uniquely identify any pixel format and are referenced by Vision Standards.

The latest list of approved 32 bit values is available to download from the GenICam web page, both as a PDF document, and as a C header file:

GenICamPixelFormatValues.pdf

PFNC.h

11 Document History

Version	Date	Editor	Description of Changes
Draft A	2010-08-06	Eric Carey, Teledyne DALSA	- Initial version from proposal to GigE Vision committee made at Yokohama technical meeting.
Draft B	2010-10-05	Eric Carey, Teledyne DALSA	- Include comments from Stephane Maurice (Matrox). - Include comments from discussion with Mike Miethig (DALSA) for Camera Link HS extensions. This lead to the addition of alignment tag and cluster tag to adequately support anticipated pixel formats for CL HS. - Add an alignment tag in the grouping section (to support CL HS). - Add cluster tag to allow regrouping of monochrome pixels before they are aligned to a given byte boundary (to support CL HS). Provide a definition for cluster. - Enforce last byte to hold the combined data for the grouped style (even though this deviates from current GEV practice).
1.0 RC1	2010-12-14	Eric Carey, Teledyne DALSA	- Add note that CFA only supports square pattern. Match description with first CFA pattern illustrated. - Clarify that YUV is used for analogue television transmission. Change examples from YUV to Y'CbCr when possible to avoid using the incorrect YUV terminology. - Used luma and chroma instead of luminance and chrominance to comply with SMPTE Engineering Guideline EG28 (Annotated Glossary of Essential Terms for Electronic Production). - Explicitly put gamma corrected value (R'G'B' and Y') in Appendix D to illustrate impact of gamma correction, as typically done in literature. - Remove Appendix A, B and C (about existing GigE Vision, CoaXPress and Camera Link HS pixel format) since this information belongs to the respective standard, not to this naming convention (was only put there for reference during the proposal review process). - Provide figure for chroma sampling. - Indicates that the YUV color conversion equations can be used for generic Y'CbCr, as the latter is more appropriate to represent digital components. - Add AIA logo and copyrights. - Change generic pixel component designation from ABCD to LMNO, as B could be confused for blue. - Adjust figure of "lsb grouped" packing style to reflect the grouped bits are sent as part of the last byte. This creates some incompatibility with a few existing pixel formats in GigE Vision specification.
1.0 RC2	2010-12-23	Eric Carey, Teledyne DALSA	- Component sequence to be separated from the rest of pixel name by an underscore. - Replace Y with L as the generic monochrome component in some example figures. - Provide explicit range of values for R', G' and B' in the color conversion equations. - Introduce r', g' and b' as reduced range from R, G and B, as per BT.601 and BT.709. - Provide the full explanation of the BT.601 and BT.709 color conversion, supporting both the 256 values full scale RGB and the 235 values scaled down 'rgb'. Depending on the input range of RGB, the proper set of color conversion equations must be used. - Add generic YCbCr in the Components section. This supplement YCbCr601 and YCbCr709.
1.0 RC3	2011-02-02	Eric Carey, Teledyne DALSA	- Clarify that Bayer_LMMN and the like represents a pixel component location and not a pixel format (section 3.1.x). - Fix typo in figure 3-6. - Adjust list of acronyms - Correction to "Grouped" packing scheme to explicitly say that first component is stored in byte 0. It was not coherent before.

			- Clarify that “lsb packed” starts the packing from lsb to msb and that “msb packed” starts the packing from msb to lsb. - Nomenclature change: cluster tag is replaced with cluster marker since cluster is not a packing style. Tag designator is now reserved for packing styles. - Provide explicit pixel construction rules for the various packing styles.
1.0 RC4	2011-05-02	Eric Carey, Teledyne DALSA	- Introduction of non-square CFA patterns. Necessary for some linescan applications. - Add section about Image and Line Boundaries. Clearly state this specification does not define which one to use. Clarify packing and grouping rules accordingly. - Allow grouped style to use more than one last byte (ex: RGB12g40).
1.0 RC5	2011-07-29	Eric Carey, Teledyne DALSA	- Use co-sited chroma alignment for sub-sampling in the sub-sampling section. - Provide the color component order when using 4:2:2 and 4:1:1 sub-sampling. - Add footnote reference to FourCC. - Create a table providing the basic color components to be used for CFA. - Add pixel format for CIELAB, CIEXYZ, HSI and HSV. - Highlight that endianess is not defined by this convention.
1.0 RC6	2011-09-02	Eric Carey, Teledyne DALSA	- Note to indicate that alpha component content can be manufacturer-specific. - Split the Bayer Location into 4 different location sequences to avoid confusion between red and blue. - Ensure lowercase/uppercase consistency in first letter of Location. - Add example to Planar mode. - Introduction of msb Unpacked to supplement the lsb Unpacked for situation where unpacked data is aligned to the msb. - Provide both co-sited and centered chroma sample positioning in Appendix 9. Indicate that co-sited is the default used by this convention. Adjust LMN422 and LMN411 accordingly. This is in-line with BT.601 and BT.709. - Line padding alignment can be different than 8-bit when specified in the referencing standard using PFNC. - Clarify that Raw format does not reference any color space.
1.0 RC7	2011-10-26	Eric Carey, Teledyne DALSA	- Various typos and grammatical improvements after proofreading. - For references, refer to a specific version of the text. - Use of lsb’s and msb’s when referring to multiple bits. - For consistency, use Align tag (not alignment tag). - Revised color space transform equations for accuracy - Add a note that no pixel format matches RGB in the range [16, 235], as defined by BT.601 and BT.709.
1.0	2011-11-01	Eric Carey, Teledyne DALSA	Official release of Pixel Format Naming Convention version 1.0 - Remove RC7 tag. - Page formatting. - Add hyperlinks to reference documents or web site.
1.1 draft A	2013-01-10	Eric Carey, Teledyne DALSA	- Creation of an appendix to list pixel format ID recommendation for 32-bit and 16-bit. This will allow different MV standards to share same ID instead of creating incompatible sets. Inclusion of values from GigE Vision 2.0, USB3 Vision 1.0 and CoaXPress 1.0. - Adjust layout using new AIA logo. - Add new pixel formats in Appendix C as per USB3 Vision 1.0 specification.

1.1.00	2013-02-01	Eric Carey, Teledyne DALSA	Official release of Pixel Format Naming Convention version 1.1 - Remove draft tag. - Page formatting and table of content generation.
1.1.01	2014-02-20	Eric Carey, Teledyne DALSA	- Change header to GenICam to match SFNC. Move document from AIA (GigE Vision) to EMVA (GenICam). No technical change.
2.0 draft A	2014-10-05	Eric Carey, Teledyne DALSA	- Improve readability of packing styles by showing direction of reading. - Add Sparse Color Filter pixel formats (pattern #1). - Clarification about optional padding bits for packed style. - Add support for 3D pixel formats. Generalization of many PFNC concept to accommodate 3D. Based on the proposal from Jan Becvar. - Introduce LM44 location. - Generalization of “color components” to “components” to allow for 3D data. - Generalization of “monochrome” to “single-component/monochrome” and “color” to “multi-components/color”. - Introduce Coord3D component designation. Add corresponding pixel format in appendix C. - Introduce floating point support based on IEC 60559. - Add clarification about dealing with line or image boundary for 3D data. - Add clarification to Raw format. - Add reference to GenICam SFNC. - Include acronyms for the A, B and C components of 3D data. - For planar mode, introduce the “plane” terminology to be equivalent to “stream”. - Move content of appendix C about Pixel Format Reference Value outside of this document. A reference is now provided in appendix C. - Introduction of bi-color format to support some 2-stage linescan color sensors. - Remove reference to GigE Vision and USB3 Vision to avoid circular dependencies.
2.0	2014-12-10	Eric Carey, Teledyne DALSA	Official release of Pixel Format Naming Convention version 2.0. - Update references to most recent version of GenICam and SFNC. - Remove draft marking. - Page formatting.
2.1 draft A	2016-01-04	Eric Carey, Teledyne DALSA	- In Packing Style section, established a default lsb ordering rule for PFNC-compliant image buffers. Camera standard using PFNC must thus define if they use lsb ordering or msb ordering by default (i.e. when lsb/msb is not explicitly spelled-out in the pixel format name). - Add a note in the Introduction section to explain how PFNC can be supplemented by camera interface standards to define bit-level encoding. This allow these standards to use PFNC naming, but with a different bit-level encoding. But when the pixel data exits the boundary of that standard, it must respect the PFNC bit-level enconding. - Revise hyperlinks. Removed reference to specific camera interface standards.
2.1	2016-02-03	Eric Carey, Teledyne DALSA	Official release of Pixel Format Naming Convention version 2.1. - Remove draft marking.
2.2 draft A	2018-08-09	Uwe Hagmaier Matrix Vision GmbH	- Added Chapter 7.2: Semiplanar Mode

2.2	2018-09-14	Uwe Hagmaier Matrix Vision GmbH	Official release of Pixel Format Naming Convention version 2.2. - Remove RC1 and “under ballot” marking
2.3 Draft 1	2018-11-18	Uwe Hagmaier Matrix Vision GmbH	Draft 1 of 2.3 – Added “Data” for GenDC
2.3 Draft 2	2019-1-18	Uwe Hagmaier Matrix Vision GmbH	Draft 2 of 2.3 – Removed “Data” for GenDC from PFNC component table, Added Subsection “Generic Data Types Formats”
2.3 Draft 3	2019-2-9	Uwe Hagmaier Matrix Vision GmbH	DRAFT 3 - Removed Note 1 below Table 3 - 3
2.3	2019-2-19	Uwe Hagmaier Matrix Vision GmbH	Removed Draft marking
2.4 Draft1	2019-9-2	Uwe Hagmaier Matrix Vision GmbH	Added GPOLxxx for polarized – first proposal
2.4 Draft2	2019-9-4	Uwe Hagmaier Matrix Vision GmbH	Added POLxxx for polarized (Option2)
2.4 Draft3	2020-10-22	Uwe Hagmaier Matrix Vision GmbH	Removed GPOL (Option 1) according to Stresa testvote Added LMN420 Location
2.4 Draft4	2021-5-10	Uwe Hagmaier Matrix Vision GmbH	Removed LMN420 Location
2.4	2021-6-16	Uwe Hagmaier Matrix Vision GmbH	Removed Draft marking Adjusted colors and notation in pol bayer diagram according to rest of the document
2.5 Draft 1	2026-2-03	Uwe Hagmaier Balluff MV GmbH	Added Proposal from Christian Hartinger, MVTec, Changed description on RGB formats with alpha component and added corresponding new RGB-formats with the newly introduced Custom-Wavelength (Cw) component
2.5 Draft 2	2026-2-20	Uwe Hagmaier Balluff MV GmbH	Included change request from #2066, Appendix C – Improve wording about how the 32 bit values are defined
2.5	2026-11-4	Uwe Hagmaier Balluff MV GmbH	Removed Draft marking Changed wordings from “computer screen” to “display device” Fixed infrared, near-infrared and ultraviolet to be proper English.