



# Pixel Format Naming Convention (PFNC)

Version 1.1

# Table of Content

1	Introduction.....	7
1.1	Purpose.....	7
1.2	Definitions and Acronyms .....	7
1.2.1	Definitions.....	7
1.2.2	Acronyms .....	7
1.3	Reference Documents .....	9
1.4	Assumptions.....	9
2	Summary of the Pixel Naming Convention .....	11
3	Components and Location .....	13
3.1	Pixel Location in Image .....	13
3.1.1	Mono Location.....	13
3.1.2	LMN444 Location .....	13
3.1.3	LMN422 Location .....	14
3.1.4	LMN411 Location .....	15
3.1.5	LMNO4444 Location.....	15
3.1.6	Bayer Location.....	16
3.1.7	CFA_XXXX Location (square pattern) .....	18
3.1.8	CFA<#lines>by<#columns>_XXXX Location (non-square pattern).....	18
3.2	Components .....	19
3.2.1	CFA Basic Components.....	21
4	Number of bits for each component.....	22
5	Optional “sign” indicator .....	23
6	Optional Packing Style .....	24
6.1	Unpacked .....	24
6.1.1	lsb Unpacked.....	24
6.1.2	msb Unpacked.....	25
6.2	Cluster marker.....	26
6.3	Packed tag.....	26
6.3.1	lsb Packed .....	26
6.3.2	msb Packed .....	28
6.4	Grouped tag.....	29

6.4.1	lsb Grouped.....	29
6.4.2	msb Grouped.....	31
6.5	Align tag.....	31
6.6	Packing Style Summary.....	33
6.7	Dealing with Line and Image Boundaries .....	34
7	Interface-specific.....	35
7.1	Planar mode .....	35
7.2	Components Sequencing.....	35
8	Appendix A - Color Space Transforms .....	36
8.1	Gamma Correction.....	36
8.2	Y'CbCr Conversions.....	37
8.2.1	Generic Full Scale Y'CbCr (8-bit).....	37
8.2.2	Y'CbCr601 (8-bit) .....	39
8.2.3	Y'CbCr709 (8-bit) .....	42
9	Appendix B - Sub-sampling notation .....	45
9.1	Co-sited Positioning.....	45
9.2	Centered Positioning.....	46
10	Appendix C – Pixel Format Value Reference.....	47
10.1	About 32-bit values.....	52
10.2	About 16-bit values.....	52
11	Document History.....	54

# List of Figures

Figure 1-1 : 8-bit pixel data ..... 9

Figure 1-2 : 16-bit pixel data ..... 9

Figure 1-3 : 32-bit pixel data ..... 9

Figure 2-1 : Naming Convention Text Fields ..... 11

Figure 3-1: Mono Pixel Location..... 13

Figure 3-2: LMN444 Pixel Location ..... 14

Figure 3-3: LMN422 Pixel Location ..... 14

Figure 3-4: LMN411 Pixel Location ..... 15

Figure 3-5: LMNO4444 Pixel Location ..... 15

Figure 3-6: BayerRG array ..... 16

Figure 3-7: Bayer\_LMMN Pixel Location ..... 16

Figure 3-8: BayerBG array ..... 16

Figure 3-9: Bayer\_NMML Pixel Location ..... 17

Figure 3-10: BayerGR array ..... 17

Figure 3-11: Bayer\_MLNM Pixel Location ..... 17

Figure 3-12: BayerGB array ..... 18

Figure 3-13: Bayer\_MNLM Pixel Location ..... 18

Figure 3-14 : Examples of 4x4 CFA from Kodak using red-green-blue-white pixels (image from Wikipedia)..... 18

Figure 3-15: CFA1by4\_GRGB array..... 19

Figure 6-1: Mono8 unpacked..... 24

Figure 6-2: Mono10 unpacked..... 24

Figure 6-3: Mono12 unpacked..... 24

Figure 6-4: Mono10msb unpacked ..... 25

Figure 6-5: Mono12msb unpacked ..... 25

Figure 6-6 : 10-bit monochrome pixel lsb packed into 32 bits (Mono10c3p32) ..... 26

Figure 6-7 : 3 components in 10-bit lsb packed into 32-bit pixel (RGB10p32) ..... 27

Figure 6-8 : 3 components lsb packed into 16-bit pixel (RGB565p)..... 27

Figure 6-9 : 10-bit monochrome pixel lsb packed (Mono10p)..... 27

Figure 6-10 : 3 components in 10-bit msb packed into 32-bit pixel (RGB10p32msb) ..... 28

Figure 6-11 : 10-bit monochrome pixel msb packed (Mono10pmsb) ..... 28

Figure 6-12: 2 monochrome 10-bit pixels with lsb grouped into 12 bits (Mono10g12) ..... 29

Figure 6-13: 2 monochrome 12-bit pixels with lsb grouped into 24 bits (Mono12g) ..... 30

Figure 6-14: 3 components of 10-bit with lsb grouped into 32-bit pixel (RGB10g32)..... 30

Figure 6-15 : 3 components of 12-bit with lsb grouped into 40-bit pixel (RGB12g40)..... 30

Figure 6-16 : 3 components of 10-bit with msb grouped into 32-bit pixel (RGB10g32msb) ..... 31

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

Figure 6-18 : Using a cluster marker of 3 unpacked Mono10 aligned to 64 bits (Mono10c3a64)..... 32

Figure 7-1: RGB10\_Planar ..... 35

Figure 8-1: Gamma Correction for ITU-R BT.601 (image from Wikipedia)..... 36

Figure 8-2 : Generic full scale Y'CbCr..... 38

Figure 8-3 : Full scale RGB for BT.601 ..... 40

Figure 8-4 : Scaled down rgb for BT.601 ..... 41

Figure 8-5 : Full scale RGB for BT.709 ..... 42

Figure 8-6 : Scaled down rgb for BT.709 ..... 43

Figure 9-1 : Chroma positioning (co-sited alignment) ..... 45

Figure 9-2 : Chroma positioning (centered alignment)..... 46

## List of Equations

Equation 1 : Gamma Correction .....	36
Equation 2 : Generic full scale R'G'B' to Y'CbCr conversion (8 bits).....	38
Equation 3 : Generic full scale Y'CbCr to R'G'B' conversion (8 bits).....	39
Equation 4 : Full scale R'G'B' to Y'CbCr601 conversion (8 bits) .....	40
Equation 5 : Full scale Y'CbCr601 to R'G'B' conversion (8 bits) .....	40
Equation 6 : Scaled down r'g'b' to Y'CbCr601 conversion (8 bits).....	41
Equation 7 : Y'CbCr601 to r'g'b' conversion (8 bits).....	42
Equation 8 : Full scale R'G'B' to Y'CbCr709 conversion (8 bits) .....	43
Equation 9 : Full scale Y'CbCr601 to R'G'B' conversion (8 bits) .....	43
Equation 10 : Scaled down r'g'b' to Y'CbCr709 conversion (8 bits) .....	44
Equation 11 : Y'CbCr709 to R'G'B' conversion (8 bits).....	44

# 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. 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.

---

Request for clarifications or to add new pixel formats should be directed at the most recent editor of the document, as listed in the Document History section.

## 1.2 Definitions and Acronyms

### 1.2.1 Definitions

<b>Cluster</b>	A group of monochrome pixels combined together and treated as a multi-component pixel.
<b>Component</b>	One of the color constituents necessary to uniquely represent a pixel color. For monochrome pixels, only one component is necessary (ex: luma). For color pixels, 3 or 4 components might be needed (ex: red-green-blue or Y'CbCr).
<b>Pixel</b>	A single point in an image that can contain more than one color component.
<b>Bayer</b>	A specific type of color filter array using a 2x2 tile with 1 red, 2 green and 1 blue components.

### 1.2.2 Acronyms

<b><i>a</i></b>	Alpha component
<b>AIA</b>	Automated Imaging Association
<b><i>b</i></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><i>B</i></b>	Full scale blue color component (ex: 256 values in 8-bit)

<b><i>Cb</i></b>	Chroma blue
<b><i>CFA</i></b>	Color Filter Array
<b><i>Cr</i></b>	Chroma red
<b><i>FourCC</i></b>	Four-Character Code
<b><i>g</i></b>	Scaled down green color component (ex: 235 values in 8-bit, must be specified by the standard referencing the Pixel Format Naming Convention)
<b><i>G</i></b>	Full scale green color component (ex: 256 values in 8-bit)
<b><i>HDTV</i></b>	High Definition Television
<b><i>Ir</i></b>	Infrared color component
<b><i>IR</i></b>	Infrared
<b><i>ITU</i></b>	International Telecommunication Union
<b><i>JIA</i></b>	Japanese Industrial Imaging Association
<b><i>JPEG</i></b>	Joint Photographic Experts Group
<b><i>L</i></b>	Generic first component of a pixel
<b><i>lsb</i></b>	Least significant bit
<b><i>LSB</i></b>	Least Significant Byte
<b><i>M</i></b>	Generic second component of a pixel
<b><i>MPEG</i></b>	Moving Picture Experts Group
<b><i>msb</i></b>	Most significant bit
<b><i>MSB</i></b>	Most Significant Byte
<b><i>N</i></b>	Generic third component of a pixel
<b><i>O</i></b>	Generic fourth component of a pixel
<b><i>r</i></b>	Scaled down red color component (ex: 235 values in 8-bit, must be specified by the standard referencing the Pixel Format Naming Convention)
<b><i>R</i></b>	Full scale red color component (ex: 256 values in 8-bit)
<b><i>SDTV</i></b>	Standard Definition Television
<b><i>Y'</i></b>	Luma
<b><i>U</i></b>	1 <sup>st</sup> chroma in YUV (blue – luma color difference)
<b><i>V</i></b>	2 <sup>nd</sup> chroma in YUV (red – luma color difference)
<b><i>W</i></b>	White color component (equivalent to monochrome)

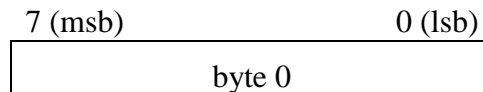


### 1.3 Reference Documents

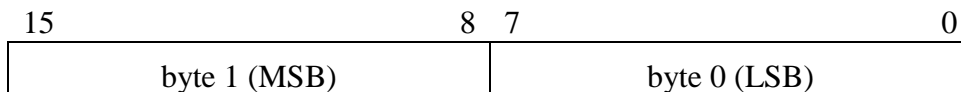
<a href="#">CoaXPress</a>	CoaXPress standard, JIA NIF-001-2010, version 1.0
<a href="#">GenICam</a>	Generic Interface for Camera, version 2.2.0
<a href="#">GigE Vision</a>	GigE Vision Video Streaming and Device Control over Ethernet Standard, AIA, version 2.0
<a href="#">ITU-R BT.601-7</a>	Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios
<a href="#">ITU-R BT.709-5</a>	Parameter values for the HDTV standards for production and international programme exchange
<a href="#">JFIF</a>	JPEG File Interchange Format, version 1.02
<a href="#">USB3 Vision</a>	USB3 Vision standard, AIA, version 1.0

### 1.4 Assumptions

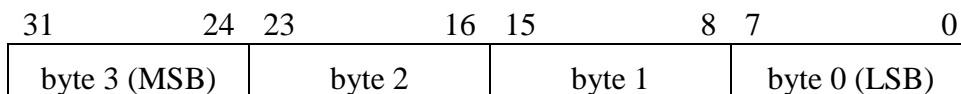
- Pixels have a maximum of 4 components (ex: alpha-red-green-blue). In this text, we use the generic LMNO designation to represent those components (ex: LMN could represent RGB where R = L, G = M and B = N).
- Some components might be sub-sampled (ex: Y'CbCr 4:2:2 and 4:1:1).
- The following figure illustrates 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 monochrome pixels put together and considered as one unit for alignment purpose. This can be used to align 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 color pixel packing concepts for a group of 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	[signed]	[packing]	[interface-specific]
-----------------------	--------	----------	-----------	----------------------

Figure 2-1 : Naming Convention Text Fields

Table 2-1 : Naming Convention Text Fields

Field	Description
<b>Components and Location</b>	Provides the list of components (ex: RGB, Y'CbCr, ...) 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).
<b># bits</b>	# of bits of each component
<b>Signed</b> (optional)	Sign indicator <ul style="list-style-type: none"> <li>• <i>empty</i> or 'u': unsigned data</li> <li>• 's': 2's complement signed data</li> </ul>
<b>Packing</b> (optional)	Packing style indicator showing how data is put into bytes and how to align them. <ul style="list-style-type: none"> <li>• <i>empty</i>: unpacked data. Empty bits of each component must be padded with 0 to align to byte boundary.</li> <li>• 'p': packed data with no bit left in between components.</li> <li>• 'g': grouped data where least significant bits or most significant bits of the components are grouped in a separate byte.</li> <li>• 'c': cluster of monochrome pixels indicating the number of pixels to put together. This marker does not provide packing information per say.</li> <li>• 'a': an additional tag indicating the pixel is aligned to the given number of bits.</li> </ul>
<b>Interface-specific</b> (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

Components provide the list of color components available in the pixel format. Location offers additional information regarding the positioning of the color 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 color 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 color format where ‘L’ represents the first color 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 color component is separated onto a different stream.

Ex: Mono8

	1	2	3	4	...
1	L <sub>11</sub>	L <sub>12</sub>	L <sub>13</sub>	L <sub>14</sub>	...
2	L <sub>21</sub>	L <sub>22</sub>	L <sub>23</sub>	L <sub>24</sub>	...
3	L <sub>31</sub>	L <sub>32</sub>	L <sub>33</sub>	L <sub>34</sub>	...
4	L <sub>41</sub>	L <sub>42</sub>	L <sub>43</sub>	L <sub>44</sub>	...
...	...	...	...	...	...

Figure 3-1: Mono Pixel Location

##### 3.1.2 LMN444 Location

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

Ex: RGB8

	1	2	3	4	...
1	LMN <sub>11</sub>	LMN <sub>12</sub>	LMN <sub>13</sub>	LMN <sub>14</sub>	...
2	LMN <sub>21</sub>	LMN <sub>22</sub>	LMN <sub>23</sub>	LMN <sub>24</sub>	...
3	LMN <sub>31</sub>	LMN <sub>32</sub>	LMN <sub>33</sub>	LMN <sub>34</sub>	...
4	LMN <sub>41</sub>	LMN <sub>42</sub>	LMN <sub>43</sub>	LMN <sub>44</sub>	...
...	...	...	...	...	...

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-color component color space. The M and N color 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 <sub>11</sub>	L <sub>12</sub>	LMN <sub>13</sub>	L <sub>14</sub>	...
2	LMN <sub>21</sub>	L <sub>22</sub>	LMN <sub>23</sub>	L <sub>24</sub>	...
3	LMN <sub>31</sub>	L <sub>32</sub>	LMN <sub>33</sub>	L <sub>34</sub>	...
4	LMN <sub>41</sub>	L <sub>42</sub>	LMN <sub>43</sub>	L <sub>44</sub>	...
...	...	...	...	...	...

Figure 3-3: LMN422 Pixel Location

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

L<sub>11</sub> , M<sub>11</sub> , L<sub>12</sub> , N<sub>11</sub> , L<sub>13</sub> , M<sub>13</sub> , L<sub>14</sub> , N<sub>13</sub> ...

The above component order is equivalent to FourCC<sup>1</sup> YUY2.

<sup>1</sup> 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-color component color space. The M and N color 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 <sub>11</sub>	L <sub>12</sub>	L <sub>13</sub>	L <sub>14</sub>	LMN <sub>15</sub>	...
2	LMN <sub>21</sub>	L <sub>22</sub>	L <sub>23</sub>	L <sub>24</sub>	LMN <sub>25</sub>	...
3	LMN <sub>31</sub>	L <sub>32</sub>	L <sub>33</sub>	L <sub>34</sub>	LMN <sub>35</sub>	...
4	LMN <sub>41</sub>	L <sub>42</sub>	L <sub>43</sub>	L <sub>44</sub>	LMN <sub>45</sub>	...
...	...	...	...	...	...	...

Figure 3-4: LMN411 Pixel Location

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

L<sub>11</sub> , L<sub>12</sub> , M<sub>11</sub> , L<sub>13</sub> , L<sub>14</sub> , N<sub>11</sub> , L<sub>15</sub> , L<sub>16</sub> , M<sub>15</sub> , L<sub>17</sub> , L<sub>18</sub> , N<sub>15</sub> ...

### 3.1.5 LMNO4444 Location

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

Ex: aRGB8

	1	2	3	4	...
1	LMNO <sub>11</sub>	LMNO <sub>12</sub>	LMNO <sub>13</sub>	LMNO <sub>14</sub>	...
2	LMNO <sub>21</sub>	LMNO <sub>22</sub>	LMNO <sub>23</sub>	LMNO <sub>24</sub>	...
3	LMNO <sub>31</sub>	LMNO <sub>32</sub>	LMNO <sub>33</sub>	LMNO <sub>34</sub>	...
4	LMNO <sub>41</sub>	LMNO <sub>42</sub>	LMNO <sub>43</sub>	LMNO <sub>44</sub>	...
...	...	...	...	...	...

Figure 3-5: LMNO4444 Pixel Location

### 3.1.6 Bayer Location

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

#### 3.1.6.1 Bayer\_LMMN Location

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

Ex: BayerRG8

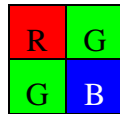


Figure 3-6: BayerRG array

	1	2	3	4	...
1	L <sub>11</sub>	M <sub>12</sub>	L <sub>13</sub>	M <sub>14</sub>	...
2	M <sub>21</sub>	N <sub>22</sub>	M <sub>23</sub>	N <sub>24</sub>	...
3	L <sub>31</sub>	M <sub>32</sub>	L <sub>33</sub>	M <sub>34</sub>	...
4	M <sub>41</sub>	N <sub>42</sub>	M <sub>43</sub>	N <sub>44</sub>	...
...	...	...	...	...	...

Figure 3-7: Bayer\_LMMN Pixel Location

#### 3.1.6.2 Bayer\_NMML Location

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

Ex: BayerBG8

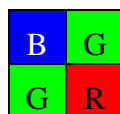


Figure 3-8: BayerBG array



	1	2	3	4	...
1	N <sub>11</sub>	M <sub>12</sub>	N <sub>13</sub>	M <sub>14</sub>	...
2	M <sub>21</sub>	L <sub>22</sub>	M <sub>23</sub>	L <sub>24</sub>	...
3	N <sub>31</sub>	M <sub>32</sub>	N <sub>33</sub>	M <sub>34</sub>	...
4	M <sub>41</sub>	L <sub>42</sub>	M <sub>43</sub>	L <sub>44</sub>	...
...	...	...	...	...	...

Figure 3-9: Bayer\_NMML Pixel Location

### 3.1.6.3 Bayer\_MLNM Location

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

Ex: BayerGR8

G	R
B	G

Figure 3-10: BayerGR array

	1	2	3	4	...
1	M <sub>11</sub>	L <sub>12</sub>	M <sub>13</sub>	L <sub>14</sub>	...
2	N <sub>21</sub>	M <sub>22</sub>	N <sub>23</sub>	M <sub>24</sub>	...
3	M <sub>31</sub>	L <sub>32</sub>	M <sub>33</sub>	L <sub>34</sub>	...
4	N <sub>41</sub>	M <sub>42</sub>	N <sub>43</sub>	M <sub>44</sub>	...
...	...	...	...	...	...

Figure 3-11: Bayer\_MLNM Pixel Location

### 3.1.6.4 Bayer\_MNLM Location

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

Ex: BayerGB8

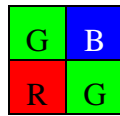


Figure 3-12: BayerGB array

	1	2	3	4	...
1	M <sub>11</sub>	N <sub>12</sub>	M <sub>13</sub>	N <sub>14</sub>	...
2	L <sub>21</sub>	M <sub>22</sub>	L <sub>23</sub>	M <sub>24</sub>	...
3	M <sub>31</sub>	N <sub>32</sub>	M <sub>33</sub>	N <sub>34</sub>	...
4	L <sub>41</sub>	M <sub>42</sub>	L <sub>43</sub>	M <sub>44</sub>	...
...	...	...	...	...	...

Figure 3-13: Bayer\_MNLM Pixel Location

### 3.1.7 CFA\_XXXX Location (square pattern)

CFA stands for generic “Color Filter Array”. It is used for CFAs other than the popular Bayer tile. ‘xxxx’ explicitly represents the sequence of color components in the square pattern expressed in raster-scan. For example, “CFA\_RBG” 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\_WBWGBWGWWGWRGWRW” is the sequence represented by the first pattern below.



Figure 3-14 : Examples of 4x4 CFA from Kodak using red-green-blue-white pixels  
(image from Wikipedia)

---

**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.

---

### 3.1.8 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 3-15: CFA1by4\_GRGB array

### 3.2 Components

The components provide the color 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
“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
“aRGB”	LMNO4444 location	alpha-Red-Green-Blue alpha component content is manufacturer-specific.
“YRGB”	LMNO4444 location	Luma-Red-Green-Blue
“RGBa”	LMNO4444 location	Red-Green-Blue-alpha alpha component content is manufacturer-specific.
“aBGR”	LMNO4444 location	alpha-Blue-Green-Red alpha component content is manufacturer-specific.
“BGRa”	LMNO4444 location	Blue-Green-Red-alpha alpha component content is manufacturer-specific.

<p>“YUV” “YUV422” “YUV411”</p>	<p>LMN444 location LMN422 location LMN411 location</p>	<p>YUV color space, typically an incorrect usage of the Y’CbCr color space. Legacy from IIDC standard.  <i>Default:</i> Y is unsigned, U and V are signed (shifted by adding 128 for 8-bit components)</p>
<p>“YCbCr” “YCbCr422” “YCbCr411”</p>	<p>LMN444 location LMN422 location LMN411 location</p>	<p>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).</p>
<p>“YCbCr601” “YCbCr601_422” “YCbCr601_411”</p>	<p>LMN444 location LMN422 location LMN411 location</p>	<p>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).</p>
<p>“YCbCr709” “YCbCr709_422” “YCbCr709_411”</p>	<p>LMN444 location LMN422 location LMN411 location</p>	<p>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).</p>
<p>“CIELAB”</p>	<p>LMN444 location</p>	<p>CIE 1976 L* a* b* color space</p>
<p>“CIEXYZ”</p>	<p>LMN444 location</p>	<p>CIE 1931 XYZ color space</p>
<p>“HSI”</p>	<p>LMN444 location</p>	<p>Hue, Saturation, Intensity</p>
<p>“HSV”</p>	<p>LMN444 location</p>	<p>Hue, Saturation, Value</p>

*Note 1:* In the above table, all components are unsigned unless explicitly stated.

*Note 2:* 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.
“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

## 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.

Use one number if all components of the pixel have same number of bits (ex: Mono8), otherwise one must list the 3 (or 4) components successively with no space in-between using 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 color components multiplied by the number of bits:

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

When the color 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 “sign” indicator

“s” if data is signed, omitted otherwise

<i>Empty</i> or “u”	Default for most component. Unsigned integer data.
“s”	Signed integer data (two’s complement).
“f” (reserved for future)	<i>Reserved</i> . Future indicator for floating point.

Use one value if all components have same sign, otherwise must list the 3 (or 4) components sign 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.

### 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.

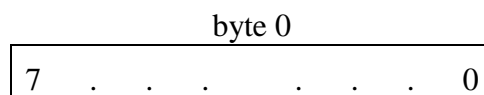
For lsb unpacked, each component is aligned to the lsb and its msb’s are zero-padded to nearest byte (8-bit) boundary. Hence next component (or pixel) always starts on the next byte. It is the typical pixel format used for image buffers on the PC-side to facilitate image processing.

---

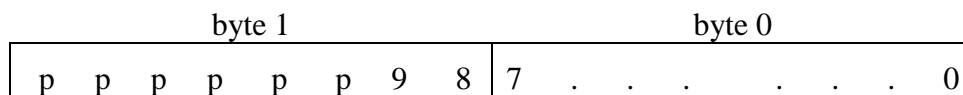
**Note:** In the following figures, the ‘p’ stands for padding bit. This means that position is a padding zero.

---

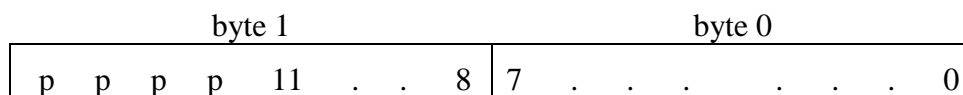
Note that in the following figures, we put **byte 0 on the right** to help illustrate the concept.



*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

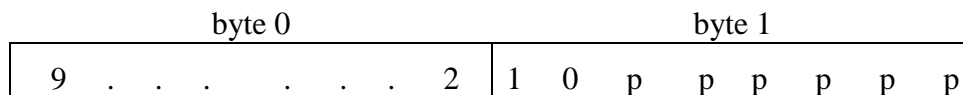
For msb unpacked, each component is filled msb first and its lsb’s are zero-padded to the nearest byte (8-bit) boundary. Hence next component (or pixel) always starts on the next byte. msb unpacked must be explicitly specified in the pixel format name by appending “msb”.

---

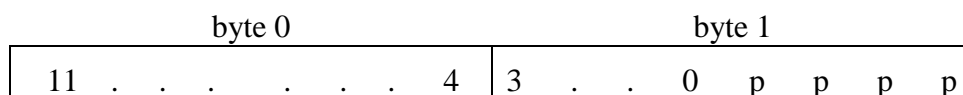
**Note:** If the component size is a multiple of 8 bits, then use lsb unpacked since no padding bits is necessary and this convention aims for the shortest string to represent the pixel name.

---

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 monochrome pixel formats. It is used to regroup a given number of monochrome pixels into one multi-component pixel. This facilitates the re-use of some color pixel packing style concepts for monochrome pixels, as most color pixels are inherently multi-components.

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

Ex: c2 = 2 monochrome pixels in the cluster

c3 = 3 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 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 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.

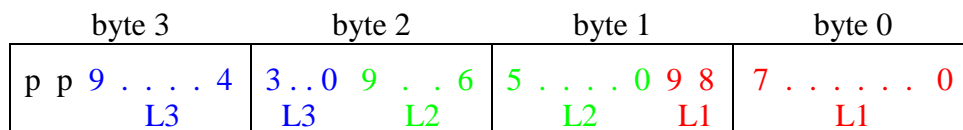


Figure 6-6 : 10-bit monochrome pixel lsb packed into 32 bits (Mono10c3p32)

## 6.3 Packed tag

**Packed** (“p”) is a common packing style where there is no bit spacing left between components (or 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 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.

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).

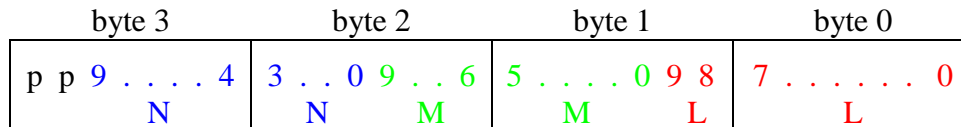


Figure 6-7 :3 components in 10-bit lsb packed into 32-bit pixel (RGB10p32)

Notice that bits are put successively for each color components with no spacing in-between.

Here is another example typical for RGB565 lsb packed:

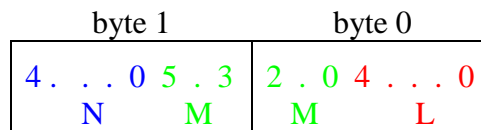


Figure 6-8 :3 components lsb packed into 16-bit pixel (RGB565p)

The following example shows how a 10-bit monochrome data can be packed from its lsb.

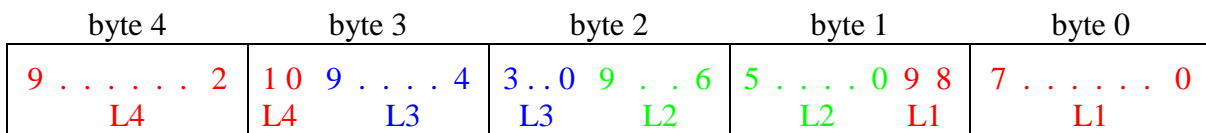


Figure 6-9 : 10-bit monochrome pixel lsb packed (Mono10p)

### 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.

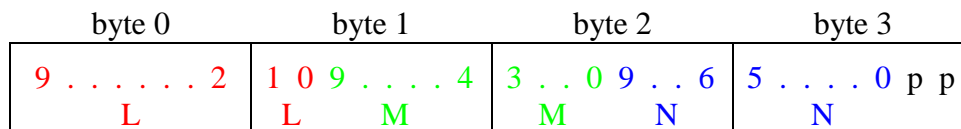
To construct the pixel stream:

- 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. This padding must consider the line or image boundary, as explained in section 6.7.

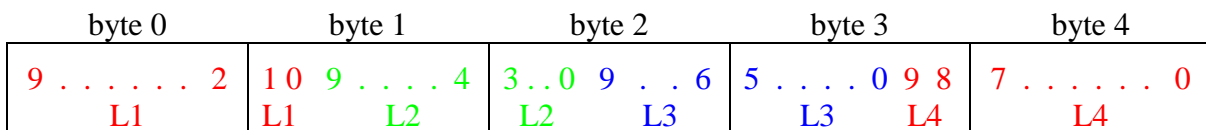
### 6.3.2 msb Packed

For msb packed, the data is filled msb first in the lowest address byte (byte 0), starting with the first component. 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.



*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. 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).

---

**Note:** There are some legacy pixel formats in GigE Vision that do not use the last byte to transmit the grouped bits. But this convention asks for any new pixel format to use the last byte for this.

---

### 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.

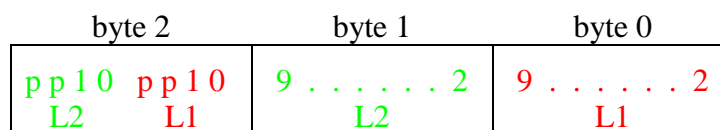


Figure 6-12: 2 monochrome 10-bit pixels with lsb grouped into 12 bits (Mono10g12)

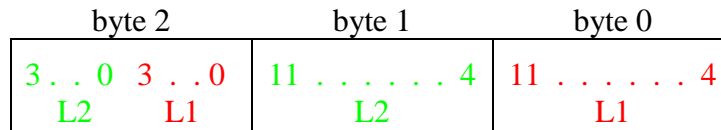


Figure 6-13: 2 monochrome 12-bit pixels with lsb grouped into 24 bits (Mono12g)

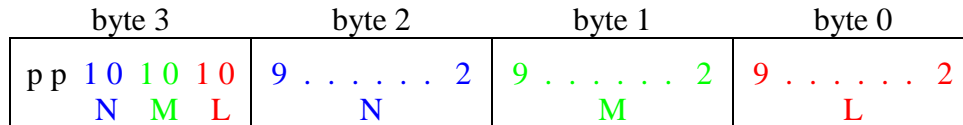


Figure 6-14: 3 components of 10-bit with lsb grouped into 32-bit pixel (RGB10g32)

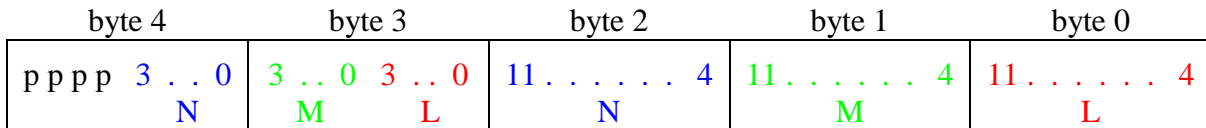


Figure 6-15 : 3 components of 12-bit with lsb grouped into 40-bit pixel (RGB12g40)

### 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. 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.

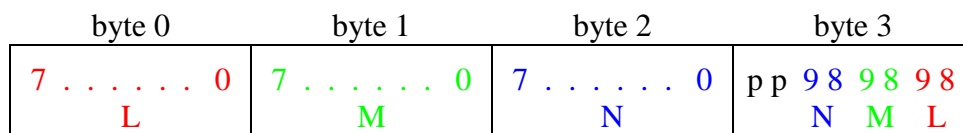


Figure 6-16 : 3 components of 10-bit with msb grouped into 32-bit pixel (RGB10g32msb)

#### 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 color components) or cluster (if the packing style refers to packing of 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.

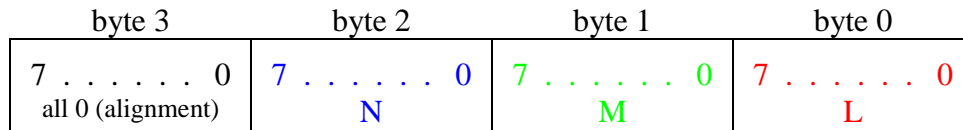


Figure 6-17: RGB 8-bit unpacked aligned to 32-bit (RGB8a32)

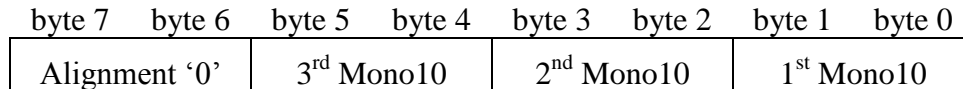


Figure 6-18 : Using a cluster marker of 3 unpacked Mono10 aligned to 64 bits (Mono10c3a64)

### 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

<i>Unpacked</i>	Most significant bits of each component are padded with zeros to be 8-bit aligned.
<i>Unpacked msb</i>	Least significant bits of each component are padded with zeros to be 8-bit aligned.
“c<x>”	Cluster of ‘x’ 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 monochrome pixels that must be aligned together, it is <b>mandatory</b> 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 exist:

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 define in their document which method is used.

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.

---

## 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 color component is transmitted over a different stream or as a separate 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

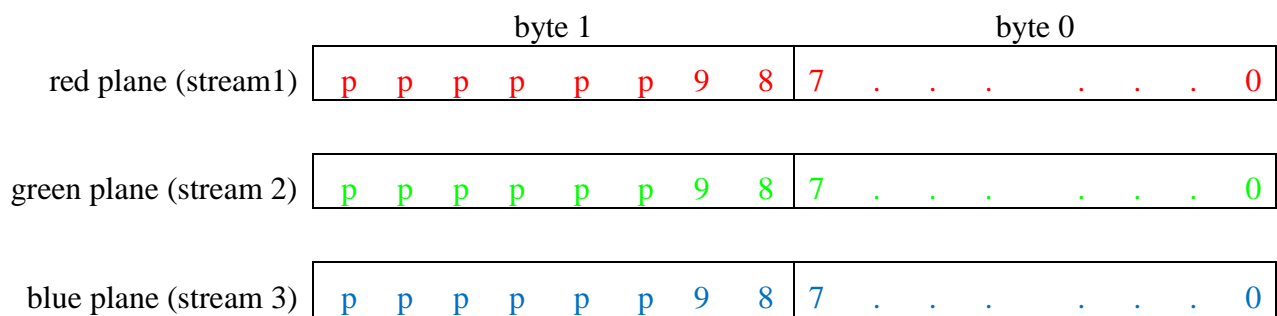


Figure 7-1: RGB10\_Planar

### 7.2 Components Sequencing

If the color 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 color 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 color components are transmitted.

In this case, an 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.

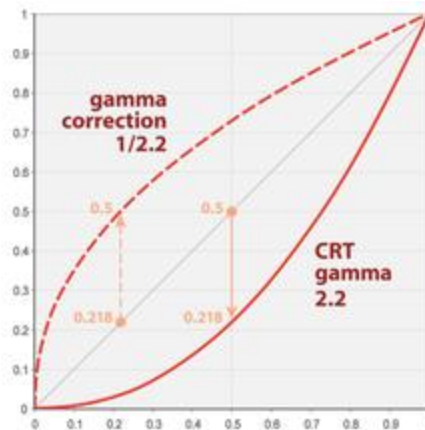
$$R' = R^{1/\gamma}$$

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

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

where  $\gamma$  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 spanning the range [0.0, 1.0].

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

$$\begin{aligned} E'_B - E'_Y \\ E'_R - E'_Y \end{aligned} \quad (3)$$

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).

$$\begin{aligned} R', G' \text{ and } B' \text{ in the range } [0, 255] \text{ (8-bit)} \\ r', g' \text{ and } b' \text{ in the range } [16, 235] \text{ (8-bit)} \end{aligned} \quad (4)$$

### 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.

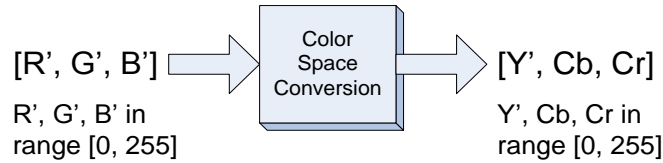


Figure 8-2 : Generic full scale Y'CbCr

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

$$E'_B - E'_Y \quad \text{is in the range } [-0.886, +0.886] \quad (5)$$

$$E'_R - E'_Y \quad \text{is in the range } [-0.701, +0.701]$$

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

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

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

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:

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

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

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

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

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

$$G' = 255 E'_G$$

$$B' = 255 E'_B$$

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:

$$\begin{aligned}
 R' &= Y' && + 1.40200 (Cr - 128) \\
 G' &= Y' && - 0.34414 (Cb - 128) - 0.71414 (Cr - 128) \\
 B' &= Y' && + 1.77200 (Cb - 128)
 \end{aligned}$$

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:

$$Y' \text{ in the range } [16, 235] \tag{9}$$

$$Cb \text{ and } Cr \text{ in the range } [16, 240]$$

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:

$$Y'_{601} = 219 E'_Y + 16 \tag{10}$$

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

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

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.

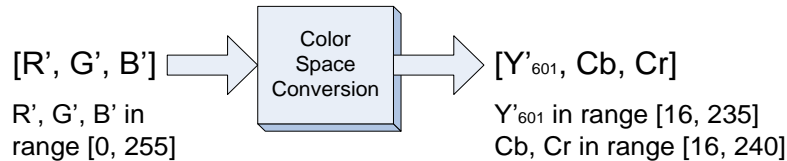


Figure 8-3 : Full scale RGB for BT.601

Replacing (8) into (10) leads to:

$$\begin{aligned}
 Y'_{601} &= 0.25679 R' + 0.50413 G' + 0.09791 B' + 16 \\
 Cb &= -0.14822 R' - 0.29099 G' + 0.43922 B' + 128 \\
 Cr &= 0.43922 R' - 0.36779 G' - 0.07143 B' + 128
 \end{aligned}$$

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].

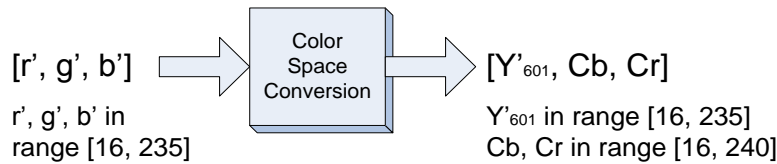
Equation 5 : Full scale Y'CbCr601 to R'G'B' conversion (8 bits)



## 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:

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

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:

$$r' = Y'_{601} + 1.37071 (Cr - 128)$$

$$g' = Y'_{601} - 0.33645 (Cb - 128) - 0.69820 (Cr - 128)$$

$$b' = Y'_{601} + 1.73245 (Cb - 128)$$

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:

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

$$E'_R - E'_Y \quad \text{is in the range } [-0.7874, +0.7874]$$

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

$$E'_{Cb} = (0.5 / 0.9278) (E'_B - E'_Y) \quad (13)$$

$$E'_{Cr} = (0.5 / 0.7874) (E'_R - E'_Y)$$

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

$$Y'_{709} = 219 E'_Y + 16 \quad (14)$$

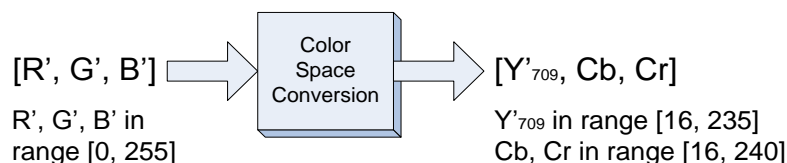
$$Cb = 224 E'_{Cb} + 128 = 224 \times (0.5 / 0.9278) (E'_B - E'_Y) + 128$$

$$Cr = 224 E'_{Cr} + 128 = 224 \times (0.5 / 0.7874) (E'_R - E'_Y) + 128$$

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)*

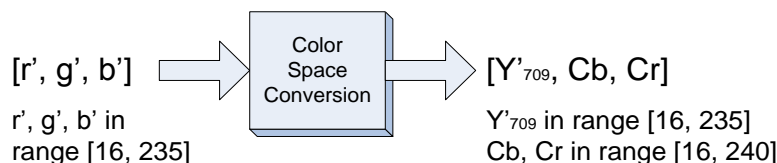
## Scaled down rgb

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.

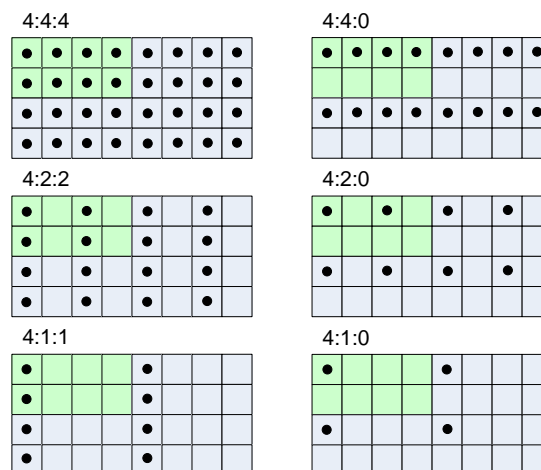


Figure 9-1 : Chroma positioning (co-sited alignment)

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.

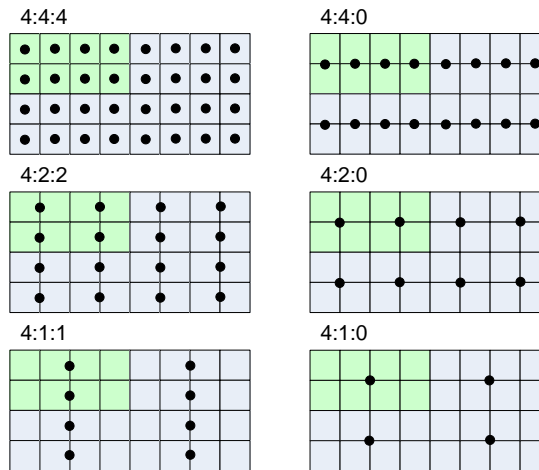


Figure 9-2 : Chroma positioning (centered alignment)

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

## 10 Appendix C – Pixel Format Value Reference

GenICam Standard Feature Naming Convention (SFNC) defines a PixelFormat feature that is typically mandatory for Machine Vision camera standards using GenICam. Many Machine Vision camera standards would typically assign a numerical value to the PixelFormat so it can be included within the image stream. This appendix provides suggested values to assign to various PixelFormat. Any new Machine Vision camera standard is invited to consider using these values to favor re-usability.

If new pixel format must be added to the table below, please contact the most recent editor of the document, as listed in the Document History section.

**Note:** The following table is only provided as reference. It is up to the standard referencing the Pixel Format Naming Convention, to indicate the values associated to each pixel format when such values are needed.

*Table 10-1 : PixelFormat Reference Values*

Pixel Format Name (alphabetical order)	32-bit value	16-bit value
BayerBG8	0x0108000B	0x0341
BayerBG10	0x0110000F	-
BayerBG10p	0x010A0052	-
BayerBG10pmsb	-	0x0342
BayerBG12	0x01100013	-
BayerBG12p	0x010C0053	-
BayerBG12pmsb	-	0x0343
BayerBG14pmsb	-	0x0344
BayerBG16	0x01100031	0x0345
BayerGB8	0x0108000A	0x0331
BayerGB10	0x0110000E	-
BayerGB10p	0x010A0054	-
BayerGB10pmsb	-	0x0332
BayerGB12	0x01100012	-
BayerGB12p	0x010C0055	-
BayerGB12pmsb	-	0x0333
BayerGB14pmsb	-	0x0334
BayerGB16	0x01100030	0x0335
BayerGR8	0x01080008	0x0311
BayerGR10	0x0110000C	-
BayerGR10p	0x010A0056	-

Pixel Format Name (alphabetical order)	32-bit value	16-bit value
BayerGR10pmsb	-	0x0312
BayerGR12	0x01100010	-
BayerGR12p	0x010C0057	-
BayerGR12pmsb	-	0x0313
BayerGR14pmsb	-	0x0314
BayerGR16	0x0110002E	0x0315
BayerRG8	0x01080009	0x0321
BayerRG10	0x0110000D	-
BayerRG10p	0x010A0058	-
BayerRG10pmsb	-	0x0322
BayerRG12	0x01100011	-
BayerRG12p	0x010C0059	-
BayerRG12pmsb	-	0x0323
BayerRG14pmsb	-	0x0324
BayerRG16	0x0110002F	0x0325
BGR8	0x02180015	-
BGR10	0x02300019	-
BGR10p	0x021E0048	-
BGR12	0x0230001B	-
BGR12p	0x02240049	-
BGR14	0x0230004A	-
BGR16	0x0230004B	-
BGR565p	0x02100036	-
BGRa8	0x02200017	-
BGRa10	0x0240004C	-
BGRa10p	0x0228004D	-
BGRa12	0x0240004E	-
BGRa12p	0x0230004F	-
BGRa14	0x02400050	-
BGRa16	0x02400051	-
Mono1p	0x01010037	-
Mono2p	0x01020038	-
Mono4p	0x01040039	-
Mono8	0x01080001	0x0101



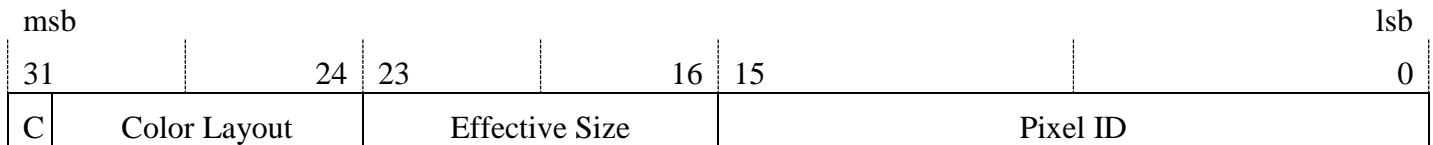
Pixel Format Name (alphabetical order)	32-bit value	16-bit value
Mono8s	0x01080002	-
Mono10	0x01100003	-
Mono10p	0x010a0046	-
Mono10pmsb	-	0x0102
Mono12	0x01100005	-
Mono12p	0x010c0047	-
Mono12pmsb	-	0x0103
Mono14	0x01100025	-
Mono14pmsb	-	0x0104
Mono16	0x01100007	0x0105
Raw* ( <i>not defined by PFNC</i> )	-	0x0000
RGB8	0x02180014	0x0401
RGB8_Planar	0x02180021	-
RGB10	0x02300018	-
RGB10p	0x021E005C	-
RGB10pmsb	-	0x0402
RGB10p32	0x0220001D	-
RGB10_Planar	0x02300022	-
RGB12	0x0230001A	-
RGB12p	0x0224005D	-
RGB12pmsb	-	0x0403
RGB12_Planar	0x02300023	-
RGB14	0x0230005E	-
RGB14pmsb	-	0x0404
RGB16	0x02300033	0x0405
RGB16_Planar	0x02300024	-
RGB565p	0x02100035	-
RGBa8	0x02200016	0x0501
RGBa10	0x0240005F	-
RGBa10p	0x02280060	-
RGBa10pmsb	-	0x0502
RGBa12	0x02400061	-
RGBa12p	0x02300062	-
RGBa12pmsb	-	0x0503

Pixel Format Name (alphabetical order)	32-bit value	16-bit value
RGBa14	0x02400063	-
RGBa14pmsb	-	0x0504
RGBa16	0x02400064	0x0505
YCbCr411_8	0x020C005A	-
YCbCr411_8_CbYYCrYY	0x020C003C	-
YCbCr422_8	0x0210003B	-
YCbCr422_8_CbYCrY	0x02100043	-
YCbCr422_10	0x02200065	-
YCbCr422_12	0x02200066	-
YCbCr601_411_8	-	0x0711
YCbCr601_411_10pmsb	-	0x0712
YCbCr601_411_12pmsb	-	0x0713
YCbCr601_411_14pmsb	-	0x0714
YCbCr601_411_16	-	0x0715
YCbCr601_411_8_CbYYCrYY	0x020C003F	-
YCbCr601_422_8	0x0210003E	0x0721
YCbCr601_422_8_CbYCrY	0x02100044	-
YCbCr601_422_10pmsb	-	0x0722
YCbCr601_422_12pmsb	-	0x0723
YCbCr601_422_14pmsb	-	0x0724
YCbCr601_422_16	-	0x0725
YCbCr601_8	-	0x0731
YCbCr601_8_CbYCr	0x0218003D	-
YCbCr601_10pmsb	-	0x0732
YCbCr601_12pmsb	-	0x0733
YCbCr601_14pmsb	-	0x0734
YCbCr601_16	-	0x0735
YCbCr709_411_8	-	0x0811
YCbCr709_411_8_CbYYCrYY	0x020C0042	-
YCbCr709_411_10pmsb	-	0x0812
YCbCr709_411_12pmsb	-	0x0813
YCbCr709_411_14pmsb	-	0x0814
YCbCr709_411_16	-	0x0815
YCbCr709_422_8	0x02100041	0x0821

Pixel Format Name (alphabetical order)	32-bit value	16-bit value
YCbCr709_422_8_CbYCrY	0x02100045	-
YCbCr709_422_10pmsb	-	0x0822
YCbCr709_422_12pmsb	-	0x0823
YCbCr709_422_14pmsb	-	0x0824
YCbCr709_422_16	-	0x0825
YCbCr709_8	-	0x0831
YCbCr709_8_CbYCr	0x02180040	-
YCbCr709_10pmsb	-	0x0832
YCbCr709_12pmsb	-	0x0833
YCbCr709_14pmsb	-	0x0834
YCbCr709_16	-	0x0835
YCbCr8	0x0218005B	-
YCbCr8_CbYCr	0x0218003A	-
YUV411_8	-	0x0611
YUV411_10pmsb	-	0x0612
YUV411_12pmsb	-	0x0613
YUV411_14pmsb	-	0x0614
YUV411_16	-	0x0615
YUV411_8_UYYVYY	0x020C001E	-
YUV422_8	0x02100032	0x0621
YUV422_10pmsb	-	0x0622
YUV422_12pmsb	-	0x0623
YUV422_14pmsb	-	0x0624
YUV422_16	-	0x0625
YUV422_8_UYVY	0x0210001F	-
YUV8	-	0x0631
YUV10pmsb	-	0x0632
YUV12pmsb	-	0x0633
YUV14pmsb	-	0x0634
YUV16	-	0x0635
YUV8_UYV	0x02180020	-

## 10.1 About 32-bit values

The 32-bit values are divided in 4 fields:



Field	Size	Description
Custom Pixel flag (C)	1 bit	Indicator for custom pixel format. 0: indicates the other fields are decoded as described by this table. 1: indicates a custom pixel format. All other fields (bit 0 to 30) are manufacturer-specific and not defined here.
Color Layout	7 bits	Provide the color layout of the pixel. Only applicable when Custom Pixel flag = 0. 0x01 = single monochrome component 0x02 = multiple color components
Effective Size	8 bits	Indicate effective number of bits occupied by the pixel (including padding). This can be used to help compute the amount of memory required to store an image. Only applicable when Custom Pixel flag = 0. Ex: 0x08 for 8-bit per pixel
Pixel ID	16 bits	A unique index to identify each pixel format. It ranges from 0x0001 to 0x0066 in this version. Only applicable when Custom Pixel flag = 0.

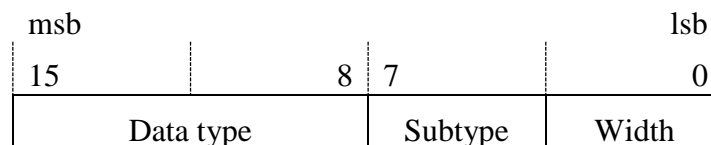
---

**Note:** The above 32-bit values are currently used by the GigE Vision and USB3 Vision camera standard sponsored by the AIA.

---

## 10.2 About 16-bit values

The 16-bit values are divided in 3 fields:



Field	Size	Description
Data type	8 bits	Indicator for the pixel type 0x01: Monochrome 0x02: Planar* (not compatible with PFNC) 0x03: Bayer 0x04: RGB 0x05: RGBa 0x06: YUV 0x07: YCbCr601 0x08: YCbCr709
Subtype	4 bits	Provide additional information to differentiate variants of “Data type”. Please consult the CoaXPress Standard for additional information about this field.
Width	4 bits	Width in bits of each color component of the pixel type. 0x1: 8 bits 0x2: 10 bits 0x3: 12 bits 0x4: 14 bits 0x5: 16 bits

---

**Note:** The above 16-bit values are currently used by the CoaXPress camera standard sponsored by the JIAA.

---

## 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	<ul style="list-style-type: none"> <li>- Include comments from Stephane Maurice (Matrox).</li> <li>- 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.</li> <li>- Add an alignment tag in the grouping section (to support CL HS).</li> <li>- 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.</li> <li>- Enforce last byte to hold the combined data for the grouped style (even though this deviates from current GEV practice).</li> </ul>
1.0 RC1	2010-12-14	Eric Carey, Teledyne DALSA	<ul style="list-style-type: none"> <li>- Add note that CFA only supports square pattern. Match description with first CFA pattern illustrated.</li> <li>- 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.</li> <li>- Used luma and chroma instead of luminance and chrominance to comply with SMPTE Engineering Guideline EG28 (Annotated Glossary of Essential Terms for Electronic Production).</li> <li>- Explicitly put gamma corrected value (R'G'B' and Y') in Appendix D to illustrate impact of gamma correction, as typically done in literature.</li> <li>- 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).</li> <li>- Provide figure for chroma sampling.</li> <li>- Indicates that the YUV color conversion equations can be used for generic Y'CbCr, as the latter is more appropriate to represent digital components.</li> <li>- Add AIA logo and copyrights.</li> <li>- Change generic pixel component designation from ABCD to LMNO, as B could be confused for blue.</li> <li>- 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.</li> </ul>
1.0 RC2	2010-12-23	Eric Carey, Teledyne DALSA	<ul style="list-style-type: none"> <li>- Component sequence to be separated from the rest of pixel name by an underscore.</li> <li>- Replace Y with L as the generic monochrome component in some example figures.</li> <li>- Provide explicit range of values for R', G' and B' in the color conversion equations.</li> <li>- Introduce r', g' and b' as reduced range from R, G and B, as per BT.601 and BT.709.</li> <li>- 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.</li> <li>- Add generic YCbCr in the Components section. This supplement YCbCr601 and YCbCr709.</li> </ul>
1.0 RC3	2011-02-02	Eric Carey, Teledyne DALSA	<ul style="list-style-type: none"> <li>- Clarify that Bayer_LMMN and the like represents a pixel component location and not a pixel format (section 3.1.x).</li> <li>- Fix typo in figure 3-6.</li> <li>- Adjust list of acronyms</li> <li>- Correction to "Grouped" packing scheme to explicitly say that first component is stored in byte 0. It was not coherent before.</li> <li>- Clarify that "lsb packed" starts the packing from lsb to msb and that "msb packed" starts the packing from msb to lsb.</li> </ul>

			<ul style="list-style-type: none"> <li>- Nomenclature change: cluster tag is replaced with cluster marker since cluster is not a packing style. Tag designator is now reserved for packing styles.</li> <li>- Provide explicit pixel construction rules for the various packing styles.</li> </ul>
1.0 RC4	2011-05-02	Eric Carey, Teledyne DALSA	<ul style="list-style-type: none"> <li>- Introduction of non-square CFA patterns. Necessary for some linescan applications.</li> <li>- Add section about Image and Line Boundaries. Clearly state this specification does not define which one to use. Clarify packing and grouping rules accordingly.</li> <li>- Allow grouped style to use more than one last byte (ex: RGB12g40).</li> </ul>
1.0 RC5	2011-07-29	Eric Carey, Teledyne DALSA	<ul style="list-style-type: none"> <li>- Use co-sited chroma alignment for sub-sampling in the sub-sampling section.</li> <li>- Provide the color component order when using 4:2:2 and 4:1:1 sub-sampling.</li> <li>- Add footnote reference to FourCC.</li> <li>- Create a table providing the basic color components to be used for CFA.</li> <li>- Add pixel format for CIELAB, CIEXYZ, HSI and HSV.</li> <li>- Highlight that endianness is not defined by this convention.</li> </ul>
1.0 RC6	2011-09-02	Eric Carey, Teledyne DALSA	<ul style="list-style-type: none"> <li>- Note to indicate that alpha component content can be manufacturer-specific.</li> <li>- Split the Bayer Location into 4 different location sequences to avoid confusion between red and blue.</li> <li>- Ensure lowercase/uppercase consistency in first letter of Location.</li> <li>- Add example to Planar mode.</li> <li>- Introduction of msb Unpacked to supplement the lsb Unpacked for situation where unpacked data is aligned to the msb.</li> <li>- 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.</li> <li>- Line padding alignment can be different than 8-bit when specified in the referencing standard using PFNC.</li> <li>- Clarify that Raw format does not reference any color space.</li> </ul>
1.0 RC7	2011-10-26	Eric Carey, Teledyne DALSA	<ul style="list-style-type: none"> <li>- Various typos and grammatical improvements after proofreading.</li> <li>- For references, refer to a specific version of the text.</li> <li>- Use of lsb's and msb's when referring to multiple bits.</li> <li>- For consistency, use Align tag (not alignment tag).</li> <li>- Revised color space transform equations for accuracy</li> <li>- Add a note that no pixel format matches RGB in the range [16, 235], as defined by BT.601 and BT.709.</li> </ul>
1.0	2011-11-01	Eric Carey, Teledyne DALSA	<p>Official release of Pixel Format Naming Convention version 1.0</p> <ul style="list-style-type: none"> <li>- Remove RC7 tag.</li> <li>- Page formatting.</li> <li>- Add hyperlinks to reference documents or web site.</li> </ul>
1.1 draft A	2013-01-10	Eric Carey, Teledyne DALSA	<ul style="list-style-type: none"> <li>- 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.</li> <li>- Adjust layout using new AIA logo.</li> <li>- Add new pixel formats in Appendix C as per USB3 Vision 1.0 specification.</li> </ul>
1.1	2013-02-01	Eric Carey, Teledyne DALSA	<p>Official release of Pixel Format Naming Convention version 1.1</p> <ul style="list-style-type: none"> <li>- Remove draft tag.</li> <li>- Page formatting and table of content generation.</li> </ul>
1.1.01	2014-02-20	<a href="#">Eric Carey</a> , Teledyne DALSA	<ul style="list-style-type: none"> <li>- Change header to GenICam to match SFNC. No technical change.</li> <li>- Move document from AIA (GigE Vision committee) to EMVA (GenICam committee).</li> </ul>

